

CogniSight SDK

User's Manual

IMAGE RECOGNITION BASED ON
NEUROMEM SILICON NETWORK

Version 5.1
Revised 06/01/2018



CogniSight SDK is a product of General Vision, Inc. (GV)

This manual is copyrighted and published by GV. All rights reserved. No parts of this work may be reproduced in any form or by any means - graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems - without the written permission of GV.

For information about ownership, copyrights, warranties and liabilities, refer to the document [Standard Terms And Conditions Of Sale](#) or contact us at www.general-vision.com.

CONTENTS

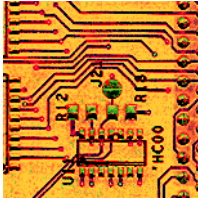
| | | |
|-------|--|----|
| 1 | Introduction..... | 4 |
| 2 | Package Content..... | 6 |
| 2.1 | Supported hardware..... | 6 |
| 2.2 | Bin Folder..... | 6 |
| 2.3 | Examples..... | 6 |
| 1. | Concepts and definitions..... | 7 |
| 2.4 | Input object types..... | 7 |
| 1. | Image Frame..... | 7 |
| 2.4.1 | Region of Interest (ROI)..... | 7 |
| 2.4.2 | Feature Extraction methods..... | 7 |
| 2.4.3 | Multiple Contexts..... | 8 |
| 2.4.4 | Region of Scan (ROS)..... | 8 |
| 2.5 | Output object types..... | 9 |
| 2.5.1 | Visual Objects (VO)..... | 9 |
| 2.5.2 | Maps..... | 9 |
| 2.5.3 | Knowledge..... | 9 |
| 2.5.4 | Project..... | 9 |
| 2.6 | Definitions..... | 10 |
| 2.7 | Dichotomy of Possible applications..... | 11 |
| 3 | Hardware Interface Functions..... | 12 |
| 3.1 | int Version();..... | 12 |
| 3.2 | int Connect(int Platform, int DeviceID);..... | 12 |
| 3.3 | int Disconnect();..... | 12 |
| 3.4 | void getNeuronsInfo(int* neuronSize, int* neuronsAvailable)..... | 12 |
| 4 | Image management..... | 13 |
| 4.1 | void BufferToCS(unsigned char *imageBuffer, int Width, int Height, int BytesPerPixel)..... | 13 |
| 4.2 | void GetCSBuffInfo(int *Width, int *Height, int *BytePerPixel)..... | 13 |
| 4.3 | void CSToBuffer(unsigned char *imageBuffer)..... | 13 |
| 5 | Region of Interest (ROI)..... | 14 |
| 5.1 | void SetROI(int Width, int Height)..... | 14 |
| 5.2 | void GetROI(int *Width, int *Height)..... | 14 |
| 5.3 | void SetFeat(int FeatID)..... | 14 |
| 5.3.1 | SubSamples..... | 14 |
| 5.3.2 | Histograms..... | 15 |
| 5.3.3 | Horizontal, Vertical or Composite Profiles..... | 15 |
| 5.3.4 | Remarks..... | 15 |
| 5.4 | void GetFeat(int *FeatID)..... | 15 |
| 5.5 | void SetFeatParams(int FeatID, int Normalize, int Minif, int Maxif, int Param1, int Param2)..... | 16 |
| 5.6 | void GetFeatParams(int *FeatID, int *Normalize, int *Minif, int *Maxif, int *Param1, int *Param2)..... | 16 |

| | | |
|--------|---|----|
| 5.7 | Int(length) GetFeature(int Left, int Top, int *Vector) | 17 |
| 5.8 | void SizeSubsample(int Width, int Height, int Monochrome, int KeepRatio);..... | 17 |
| 6 | REgions of Search (ROS)..... | 18 |
| 6.1 | void SetROS(int Left, int Top, int Width, int Height) | 18 |
| 6.2 | void GetROS(int *Left, int *Top, int *Width, int *Height)..... | 18 |
| 6.3 | Int(vectorNbr) GetROSVectors(int stepX, int stepY, unsigned char *Vectors, int *VLength) | 18 |
| 7 | Learning functions | 19 |
| 7.1 | int LearnROI(int Left, int Top, int Category) | 19 |
| 7.2 | int LearnROS(int stepX, int stepY, int category) | 19 |
| 7.3 | int BuildROSCodebook(int stepX, int stepY, int CatAllocMode)..... | 19 |
| 7.4 | int ROSToNeurons(int stepX, int stepY, int UsePositionsAsContext) | 20 |
| 8 | Recognition functions..... | 20 |
| 8.1 | Int(nsr) BestMatchROI(int Left, int Top, int *distance, int *category, int *nid)..... | 20 |
| 8.2 | int RecognizeROI(int Left, int Top, int K, int *distances, int *categories, int *nids) | 21 |
| 8.3 | int FindROSOBJECTS(int stepX, int stepY, int skipX, int skipY, int *Xpos, int *Ypos, int* distance, int* category, int* nid)..... | 21 |
| 8.4 | int FindROSAnomalies(int stepX, int stepY, int MaxNbr, int *Xpos, int *Ypos) | 22 |
| 8.5 | int MapROS(int stepX, int stepY, int *CatMap, int *DistMap, int *NidMap) | 23 |
| 9 | Knowledge Level Functions | 24 |
| 9.1 | int SaveProject(char *filename)..... | 24 |
| 9.2 | int LoadProject(char *filename)..... | 24 |
| 10 | Appendix B: Tutorial, Hints and Tips | 26 |
| 10.1 | Supervised Object Learning | 26 |
| 10.2 | Using multiple feature extractions for object classification..... | 27 |
| 10.3 | What if an object appears at different scale factors? | 28 |
| 10.4 | Dealing with objects of different sizes | 29 |
| 10.4.1 | Example 1: Learn two filling levels sampled at a same location in two consecutive images | 29 |
| 10.4.2 | Example 2: Learn two labels sampled at different X locations in a same image..... | 29 |
| 10.4.3 | Example3: Inspect the two ROIs and accept the bottle if both label and filling level are correct | 29 |
| 10.5 | Surface Inspection..... | 30 |

1 INTRODUCTION

The CogniSight SDK is a software development kit for image learning and recognition powered by a single or multiple NeuroMem neural network. It can address many applications in image, movie and video analytics and be deployed in systems ranging from smart photocell, consumer and professional cameras, industrial vision systems, etc.

Images



- Machine vision
- Surface inspection
- Medical and Satellite imaging...

Movies



- Video surveillance
- Novelty detection
- Motion analysis...

Live Video



- Target tracking
- Smart motion detection
- Stereoscopy...

LEARN



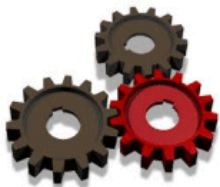
- Learn objects, textures or scenes by example using single or multiple experts
- Autonomous learning of textures for classification and anomaly detection
- Can learn raw data and signatures extracted from images, video frames, but also audio sound, MEMS, and more adding robustness to the recognition with additional contextual information
- Knowledge can be saved and restored. Tuned and enriched over time
- Build dictionaries of sparse codes autonomously to generate synthetic description of image content

RECOGNIZE



- Search and enumerate objects in an image or given regions
- Build descriptors of recognized Visual Objects
- Find novelties and anomalies in an image in an image or given regions
- Track single and multiple targets in a video sequence
- Generate meta data describing objects and textures, their locations, density, categories, and more
- Report uncertainties which is essential to improve accuracy and reduce mistakes
- Find and quantify discrepancies between an image and a reference template

TRANSFORM



- Segment an image into significant regions
- Generate maps showing the spatial distribution of objects or matching patterns per category, or confidence level
- Extract the saliency within an image
- Generate disparity map between two stereoscopic images
- Compress an image

The NeuroMem network allows to model reference objects or patterns using a library of predefined feature extractions suitable for color, shape, and texture characterization. To increase accuracy, multiple experts can be defined to classify objects based on different features and therefore complement one another and waive uncertainties if any. The final decision can be made through a rule-based decision or another feed-forward NeuroMem network trained to consolidate the responses of the multiple experts. Another unique capability of a

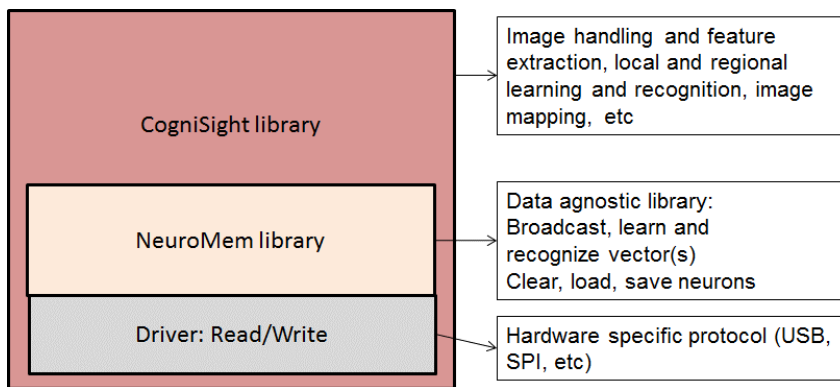
NeuroMem expert for surveillance, predictive maintenance and many other applications is to report when an object or pattern is not recognized.

The CogniSight SDK comes with a simulation of the NeuroMem network (1024 neurons per default), but to really benefit from the performance of a NeuroMem digital network it can interface to NeuroMem Smart hardware.

Benefits of the NeuroMem hardware:

- Access to a high speed Radial Basis Function (RBF) classifier or K Nearest Neighbor
- A response time which is deterministic and independent the number of models stored in the neurons
- An automatic model generator capable of learning feature vectors in real-time
- Ability to save and restore the knowledge bases
- Portability to a hardware platform with a small foot-print and low power consumption

The CogniSight SDK is developed on top of the CogniPat SDK and therefore it also includes pattern learning and recognition functions which are agnostics to data types. This means that you can extract custom features from your images, but also waveforms, text, etc. These functions are described in the [CogniPat SDK manual](#).



2 PACKAGE CONTENT

2.1 Supported hardware

The CogniSight SDK is a Dynamic Link Library which interfaces to a chain of NeuroMem neurons for learning, classification, knowledge saving and restore. Several versions of the DLL exist. They have the same entry points but include different drivers for access to the NeuroMem hardware:

CogniSight_NeuroShield Cypress USB serial driver (and simulation if board not found)
CogniSight_NSnK FTDI USB driver for NeuroStack (and simulation if board not found)
CogniSight_Simu Cycle accurate simulation of the NeuroMem network of 1024 neurons

2.2 Bin Folder

- Win32/ CogniSight_Simu.dll, CogniSight_NSnK.dll, CogniSight_NeuroShield.dll
- x64/ CogniSight_Simu.dll, CogniSight_NSnK.dll, CogniSight_NeuroShield.dll
- CogniSight.h Header defining the entry points to the DLL and specific to image handling
- CogniPat.h Header defining the entry points to the DLL and agnostic to data type
- CogniSight.cs Class defining the entry points to the DLL for C# interface
- CogniSightClass.m Class defining the entry points to the DLL for MatLab interface

2.3 Examples

Examples are provided to help understand how to use the neurons to learn and recognize visual objects or events. Depending on your hardware, each example can be executed calling one of the following DLLs:

- CogniSight_Simu.dll
- CogniSight_NeuroShield.dll
- CogniSight_NSnK.dll

| | Location of the CogniSight DLL | Location of the wrapper |
|----------------|--------------------------------|--|
| C++ | CogniSight_SDK\bin | Link CogniSight.lib to Project/Properties/Linker/Input (declared in C++ project) |
| C# | CogniSight_SDK\bin | CogniSight_SDK\CogniSight.cs |
| MatLab | CogniSight_SDK_MatLab\bin | CogniSight_SDK_MatLab\CogniSightClass.m |
| LabVIEW | CogniSight_SDK_LV\bin | CogniSight_SDK_LV\CogniSight VIs |

1. CONCEPTS AND DEFINITIONS

This paragraph describes the different types of components manipulated by a CogniSight Engine.

2.4 Input object types

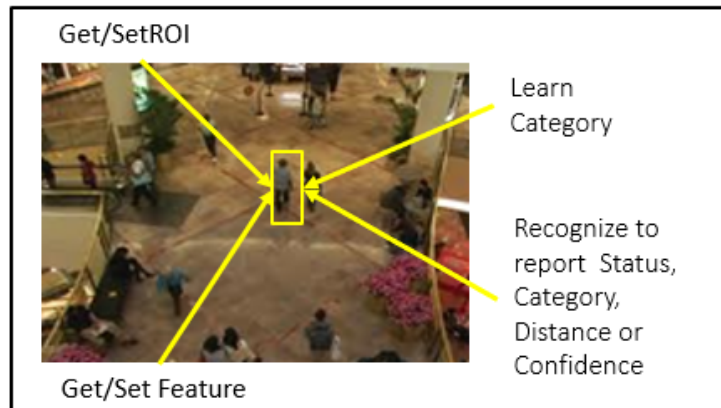
1. Image Frame

When an image is loaded from an image file or a movie file, it must be transferred to the CogniSight image memory frame so the engine can access the pixel values for feature extraction, learning and recognition. This transfer is not necessary if the image is acquired by a CogniSight Sensor in which case it resides automatically in the CogniSight image memory frame.

2.4.1 Region of Interest (ROI)

A Region Of Interest (ROI) is the primitive area to learn or recognize. It can be a discrete object, part of an object, a significant feature in a scene, a patch of texture, etc. From the pixel values inside an ROI, the CogniSight engine extracts a signature. This signature becomes the feature **vector** learned or recognized by the neurons. The CogniSight API includes a selection of pre-defined feature extractions.

The classification of the ROI can be based on a single or multiple features (or signatures). In the case of multiple features, the context of the neurons must be changed for each type of feature. The context value can be assigned to a feature identification number for example starting at the value 1 (value 0 is reserved to activate neurons of all contexts at once). For example, the feature subsample can be assigned to context 1 and the feature histogram can be assigned to context 2. The proper context must be activated prior to broadcasting the corresponding feature vector.



Also, if an application must classify more than one family of objects at a time (i.e. (1) filling level of a bottle and (2) quality of its front label), it shall store the definition of the different ROIs and make sure to change the Global Context of the neurons when switching between ROIs. For example the Context #1 will be assigned to the neurons taught with examples of filling levels and the Context #2 to the neurons taught with examples of labels. For more information about the usage of multiple ROIs refer to Appendix C.

2.4.2 Feature Extraction methods

The memory of the neurons in the NM500 and CM1K chips is 256 bytes which fits a block of 16x16 pixels without any compression. An area of 32x8 pixels will also fit without compression and might be more relevant to recognize elongated objects. If the raw data cannot fit into the memory of the neurons, a feature extraction must be applied to obtain signature fitting into the neurons' memory. Examples of features include but are not limited to sub-

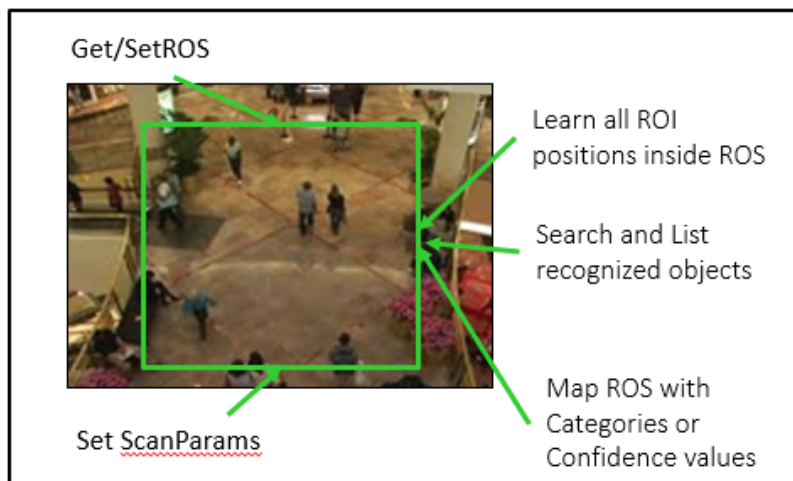
sampling, histograms of intensities, profiles, histograms of gradients, SURFS vectors, etc. The features implemented in the SDK are described under the chapter about the Region of Interest.

2.4.3 Multiple Contexts

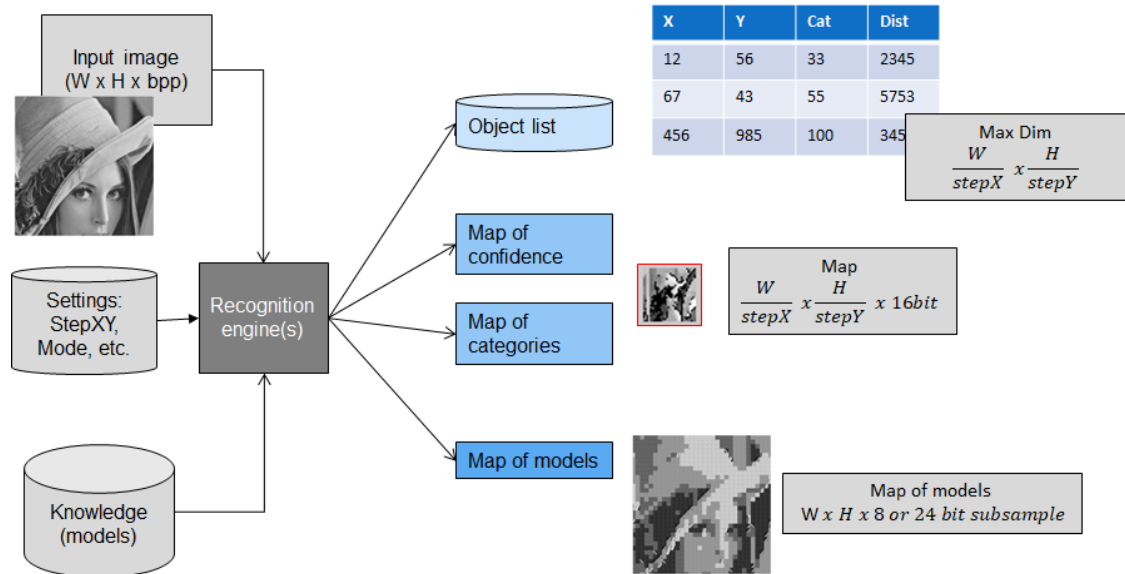
The NeuroMem neurons can be segmented dynamically into sub-networks to learn identical or different objects under different contexts. A context is user-defined and can be a type of feature, a scale at which an object is learned, a sensor type, and more. Aggregating the responses of multiple sub-networks trained differently to recognize the same inputs helps make more confident decision.

2.4.4 Region of Scan (ROS)

The Region of Scan (**ROS**) is the area to learn or recognize through the “aperture” of a given ROI and using the knowledge of the neurons. It is usually associated to scanning parameters including stepx and type of displacement.



2.5 Output object types



2.5.1 Visual Objects (VO)

The Visual Objects (**VO**) are a list of identified locations with their recognized category as a result of a Search over a region of scan. They can be presented as an array or in Transform images showing their spatial distribution based on attributes such as their category or their similarity factor.

2.5.2 Maps

A Map (**MAP**) is a transform image of a Region of Search (which can be the entire image) showing the spatial distribution of the recognized objects in term of category value, distance/confidence value or neuron identification value. The dimension of the map is the dimension of the ROS divided by the selected scanning step.

2.5.3 Knowledge

The knowledge file (**KN**) is generated by the NeuroMem network as it learns examples of visual objects or patterns extracted from the images and submitted by the CogniSight engine to the neurons. It can be tuned and enriched over time with new examples. Depending on the application, the selection of the objects to learn can be supervised or unsupervised. The knowledge can be built using single or multiple feature vectors characterizing an object or texture.

2.5.4 Project

A project file includes all the necessary information (1) to apply a knowledge for the recognition of ROS in still or live images, and (2) to expand the knowledge if necessary by learning additional examples using the correct ROIs and associated feature extractions.

2.6 Definitions

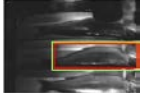



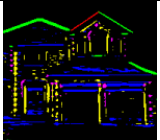
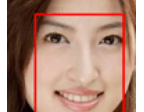


The following list describes typical variables manipulated by a CogniSight Engine

- frameWidth
- frameHeight
- frameBPP or Bit perPixels

- roiWidth
- roiHeight

- rosWidth
- rosHeight

2.7 Dichotomy of Possible applications

| Application | ROI | ROS | Decision rules | Example |
|------------------------------|----------|------------------|-------------------------------|---|
| Simple part inspection | Single | Single | Best match category |  |
| Complex part inspection | Multiple | Multiple | Decision rule on Best Matches |  |
| OCR/Kanji Character | Single | Single | Sequence of the Best Matches |  |
| Surface anomaly | Single | Single, Multiple | Unknown locations |  |
| Edge detection | Single | Single | Map of the Best Matches |  |
| Cooperative Face Recognition | Multiple | Single | Decision rule on Best Matches |  |
| Face detection in a scene | Multiple | Single | Sequence of the Best Matches |  |
| Target Tracking | Single | Single, Adaptive | Center of the Best Matches |  |

3 HARDWARE INTERFACE FUNCTIONS

3.1 int Version();

Returns the version of the DLL.

3.2 int Connect(int Platform, int DeviceID);

Establishes communication with your NeuroMem platform and if applicable a specific DeviceID. This function returns 0 if the connection is successful.

| Platform code | DeviceID |
|--|---|
| 0= Simulation of NeuroMem neurons | Reserved. Default is 0. |
| 1=NeuroStack board (4096 neurons, stackable) | Reserved. Default is 0. |
| 2= NeuroShield board (576 neurons, expandable) | Reserved. Default is 0 |
| 2= V1KU camera (1024 neurons of 256 bytes) | DeviceID is the USB device number connected to your host. Default is 0. |

The Connect function also detects the number of neurons available in the selected platform and clears their content. Its execution is necessary to size the network if the hardware allows for an expansion of the network and takes approximately 1 second per thousand neurons. The network size can then be read with the getNetworkInfo function.

3.3 int Disconnect();

Closes the communication with the current device.

3.4 void getNeuronsInfo(int* neuronSize, int* neuronsAvailable)

Read the specifications of the silicon neurons:

- neuronSize, memory capacity of each neuron in byte
- neuronsAvailable, number of neurons available

The number of neurons available is returned by the Connect function in the case of a platform with fixed capacity, and CountNeuronsandRset in the case of a platform with variable capacity.

4 IMAGE MANAGEMENT

When an image is loaded from an image file or a movie file, it must be transferred to a memory frame, which we call the CogniSight memory frame, so the CogniSight engine can access the pixel values for the feature extractions.

Depending on the application, the CogniSight memory frame can hold the source image as a whole or in part. Also, if it is known that the application does not require the use of color information for the feature extraction, the CogniSight memory frame can be limited to the grey-level pixel values and not their red, green and blue intensities.

The CogniSight memory frame supports 1 byte per pixel in the case of a monochrome plane or 3 bytes per pixel in the case of a color plane where the 3 bytes represent the succession of R, G, B intensities.

4.1 void BufferToCS(unsigned char *imageBuffer, int Width, int Height, int BytesPerPixel)

Load a byte array into a memory frame of the CogniSight workspace with a size equal to Width * Height * Bytes Per Pixel. Depending on your application, the byte array can be a full image, a particular frame in a movie file, a digitized video frame, or a user-defined region within the above mentioned.

If the BytesPerPixels value is 3, the function expects an imageBuffer with interlaced red, green, blue pixel values.

4.2 void GetCSBuffInfo(int *Width, int *Height, int *BytePerPixel)

Reads the parameters Width, Height and BytePerPixel describing the current CogniSight memory frame.

4.3 void CSToBuffer(unsigned char *imageBuffer)

Read the CogniSight memory frame and returns it as a byte array. The format of this byte array is described by the CSBuffInfo parameters Width, Height and BytePerPixel. This function can be useful to retrieve an image which was directly acquired on a compatible hardware with on-board sensor.

5 REGION OF INTEREST (ROI)

The Region Of Interest (ROI) is the primitive area to learn or recognize. It can be a discrete object, part of an object, a significant feature in a scene, a patch of texture, etc. From the pixel values inside an ROI, the CogniSight engine extracts a signature which becomes the feature **vector** learned or recognized by the neurons.

5.1 void SetROI(int Width, int Height)

Defines the nominal width and height of the current ROI.

5.2 void GetROI(int *Width, int *Height)

Reads the nominal width and height of the current ROI.

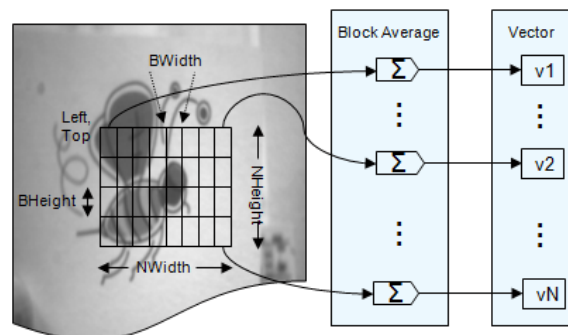
5.3 void SetFeat(int FeatID)

Set the feature extraction method as one of the following:

- 0=GreySubsample
- 1=GreyHisto
- 2=GreyHistoCumul
- 3=ColorSubsample
- 4=ColorHisto
- 5=ColorHistoCumul
- 6= Composite profile
- 7= Horizontal profile
- 8= Vertical profile

5.3.1 SubSamples

Subsample is a vector which appends the average intensity of blocks of pixels extracted from the region of interest. The blocks are all the same size, but not necessarily square. They are surveyed in a raster displacement and their average intensity is assembled into vector.



- The feature called Subsample extracts the average of the intensity value of each block regardless if the image is monochrome or color. The region must contain less than 256 blocks so the output vector fits in the 256 bytes of memory of the neurons
- The feature called SubsampleRGB extracts the average of the Red, Green and Blue intensities of each block in the ROI, so three average values per block. Therefore, the ROI must contain less than 85 blocks so the output vector fits in the 256 bytes of memory of the neurons.

5.3.2 Histograms

Histogram is a vector which gives the distribution of the grey-level values in the region of interest. It indicates the number of shades in the ROI, the presence of noisy pixels and more.

- HistoCumul or cumulative histogram is a mapping of the standard histogram which counts the cumulative number of pixels in all of the bins up to the specific bin.
- HistogramRGB is a vector which gives the distribution of the Red, Green and Blue intensities in the region of interest. It is assembled as a series of 3 histograms of 85 bins each representing 85 bins for the Red, 85 bins for the Green and 85 bins for the Blue.
- HistoCumulRGB or cumulative color histogram is a mapping of the standard histogram which counts the cumulative number of pixels in all of the bins up to the specific bin.

5.3.3 Horizontal, Vertical or Composite Profiles

The average intensity along a column or row of pixels, or both appended in a same feature vector. The composite profile can be helpful to classify the alignment of objects. The vertical and horizontal profiles to classify edges and geometric transitions.

5.3.4 Remarks

- Remark 1: The selection of the feature should not be linked to the format of the image (whether monochrome or color), but rather to the discriminating characteristic of the object, surface or scenery. For example, if your application consists of verifying the alignment of an object by detecting that its edges, the color information might be irrelevant. A grey level subsample might be sufficient to learn aligned and non aligned patterns and discarding the color information might reduce unnecessary variability to address the problem.
- Remark 2: If the image is color and the selected FeatID calls for grey level information, the feature extraction will calculate the grey level intensity of a pixel as the average of its red, green and blue intensities.
- Remark 3: If the image is monochrome and the selected FeatID calls color information, the feature extraction will replace the non-existent red, green and blue intensities with the grey-level intensity of the pixel. The resulting vector will have a profile showing a signature repeated three times, so only 1/3 of its length contains significant information.

5.4 void GetFeat(int *FeatID)

Set the feature extraction method in use.

5.5 void SetFeatParams(int FeatID, int Normalize, int Minif, int Maxif, int Param1, int Param2)

Each feature extraction method has a set of 5 attributes called FeatParams. In the present version of the SDK and considering the list of proposed features two parameters Param1 and Param2 can be used for the calculation of the feature. Refer to the table below for their description per FeatID. The amplitude of the feature vector is also affected by the option to Normalize or not. When changing FeatID and its associated parameters, it is wise to verify the values of the Minimum and Maximum Influence fields which must be related to the dimensionality of the feature vectors.

| | |
|-----------|--|
| FeatID | Identifier of the feature extraction (see table below) |
| Normalize | Flag indicating if the amplitude of the feature vector shall be normalized between [0-255] |
| Minif | Minimum influence field (for any upcoming training) |
| Maxif | Maximum influence field (for any upcoming training) |
| Param1 | 1st parameter of the feature extraction (if applicable) |
| Param2 | 2 nd parameter of the feature extraction (if applicable) |

| FeatID | Description | Param1 | Param2 |
|--------|---|----------------|-----------------|
| 0 | Monochrome subsample, or average intensity of up to 256 blocks fitting inside the ROI | Block width(*) | Block height(*) |
| 1 | Grey histogram, or the number of pixels per 256 grey-level values. The amplitude of the histogram is scaled by multiplying it by the ratio (255/N) with N, the number of pixels in the ROI. | n/a | n/a |
| 2 | Grey histogram cumulative | n/a | n/a |
| 3 | Color subsample, or average Red, Green and Blue intensities of up to 85 blocks fitting inside the ROI | Block width | Block height |
| 4 | Color histogram, or sequence of the Red, Green and Blue histograms, each of 85 values. | n/a | n/a |
| 5 | Color histogram cumulative | n/a | n/a |
| 6 | Composite profile | n/a | n/a |
| 7 | Horizontal profile | n/a | n/a |
| 8 | Vertical profile | n/a | n/a |

(*) The ROI with a size [NWidth, NHeight] is divided into blocks of size [BWIDTH, BHEIGHT].

- The pixels of block #i are averaged to produce the ith component of the signature vector.
- The relationship between the four parameters is :
 - o $NWIDTH = n * BlockWIDTH$
 - o $NHEIGHT = m * BlockHEIGHT$
 - o $n * m \leq 256$ in the case of a monochrome subsample
 - o $n * m \leq 83$ in the case of an R+G+B subsample

5.6 void GetFeatParams(int *FeatID, int *Normalize, int *Minif, int *Maxif, int *Param1, int *Param2)

Reads the current feature extraction and learning settings (see above description)

5.7 **Int(length) GetFeature(int Left, int Top, int *Vector)**

Returns the feature vector extracted from the ROI at the (X,Y) location in the image.

The type of feature is defined by the last execution of the SetFeature function.

The output vector is composed of “length” components to the neurons. Note that the function takes an array of int, but the upper byte is expected to be equal to 0 since the existing NeuroMem chips are limited to byte array memories.

5.8 **void SizeSubsample(int Width, int Height, int Monochrome, int KeepRatio);**

Calculates the parameters 0 and 1 for the extraction of the subsample, whether monochrome (featID=0) or color (featID=3). These 2 parameters are the width and heights of the internal blocks fitting inside the ROI, with the trend to fit as close to 256 internals as possible to achieve a greater resolution.

If necessary, the resulting values of these parameters can be read using the GetFeatParams function.

6 REGIONS OF SEARCH (ROS)

6.1 void SetROS(int Left, int Top, int Width, int Height)

Defines the current Region Of Search which can range from the size of the current ROI to the entire image.

This function does not verify the consistency of the input parameters and, in particular, if the ROS extends outside the image stored in the CogniSight memory buffer. The functions applying to the ROS do not verify consistency either and assume that values are correct.

6.2 void GetROS(int *Left, int *Top, int *Width, int *Height)

Reads the current Region Of Search

6.3 Int(vectorNbr) GetROSVectors(int stepX, int stepY, unsigned char *Vectors, int *VLength)

Extracts the list of feature vectors extracted from the Region of Search scanned in a raster displacement with horizontal stepX and vertical stepY. This function is useful to export feature vectors for further manipulation with the CogniPat SDK for example, or for backup and traceability purposes.

VLength reports the length of the feature vectors which is function of the featID and its parameters in use.

Vectors is an array with a dimension of VLength times Number_of_Steps covered during the scanning.

The function returns the number vectors.

7 LEARNING FUNCTIONS

7.1 int LearnROI(int Left, int Top, int Category)

Learns the feature vector extracted from the ROI at the (Left, Top) location in the image as Category. Category can range between 1 to 32766. A category of 0 can be used to teach a counter example or a background example.

The function returns the number of committed neurons (ncount). Note that this number does not increase necessarily after each execution of the Learn function. Ncount will increase only if the vector and its associated category represents novelty to the committed neurons.

Considerations before calling this function:

- Change the Global Context Register to match the context represented by the input vector
- Change the Min Influence Field and Max Influence Field
- Make sure the network is in RBF mode (Write NSR 0).
 - o The KNN mode is not appropriate for learning since it will create a new neuron each time a new category value is taught and do nothing more. The RBF mode must be used to build a decision space modeling multiple categories and also with areas of “unknown” or “uncertainties” which are essential for true artificial intelligence with voluntary redundancy for accuracy, context awareness, hypothesis generation and more.
 - o The Save and Restore mode is not compatible with the learning mode.

7.2 int LearnROS(int stepX, int stepY, int category)

Learns the feature vectors extracted from the Region of Search scanned in a raster displacement with horizontal stepX and vertical stepY. All these vectors are assigned the same user-defined category.

7.3 int BuildROSCodebook(int stepX, int stepY, int CatAllocMode)

Learns the feature vectors extracted from the Region of Search scanned in a raster displacement with horizontal stepX and vertical stepY. All the vectors representing novelty are automatically learned and assigned a category value incremented each time a novelty is detected by the neurons.

CatAllocMode: Defines which category to assign to a block which is not recognized by the currently committed neurons:

- o 0: constant value
- o 1: auto-increment by 1
- o 2: maximum delta between the vectLen components
- o 3: average value of the vectLen components
- o 4: index of the vector committing the neuron. This information can be used to retrieve the XY origin of the vector in an image, or else.

Note that the function does not clear the knowledge and uses current Minif, Maxif, GCR. It returns the number of committed neurons.

7.4 int ROSToNeurons(int stepX, int stepY, int UsePositionsAsContext)

Scan the ROS in a raster displacement and at each step load the feature vector of each ROI into the neurons assigning an incremental categories so each ROI commits a neuron. If the feature extraction is a subsample of 16x16, this means that the neurons hold the entire image divided into tiles of 16x16 pixels.

The resulting neurons have their category equal to their identifier. They are both equivalent to an encoding of the XY coordinates of the model within the ROS referential.

The function clears the existing knowledge, but restores the global registers upon termination such as the NSR, MAXIF and GCR. It returns the number of neurons holding a row of pixel data. This number depends on the image size, ROI size and Step. It is useful to convert a neuron identifier into a pixel location in the original image.

Option to use the position of the input block to encode a context. This can be useful to limit the firing patterns to vertical rows and columns in recognition.

0= uses the current context

1= set context to horizontal position of the block (modulo 127 since the context must range between [1,127])

2= set context to vertical position of the block (modulo 127 since the context must range between [1,127])

The function can be executed in LR or SRmode If SR mode, set the default Maxif to a significant value prior to execution.

8 RECOGNITION FUNCTIONS

8.1 Int(nsr) BestMatchROI(int Left, int Top, int *distance, int *category, int *nid)

Recognizes the feature vector extracted from the ROI at the (Left, Top) location in the image and reports the distance, category and neuron identifier of the closest neuron.

The function returns the Network Status Register and its lower byte can be decoded as follows:

- NSR=0, the vector is not recognized by any neuron (UNKnown)
- NSR=8, the vector is recognized and all firing neurons are in agreement with its category (IDentified)
- NSR=4, the vector is recognized but the firing neurons are in disagreement with its category (UNCertain)
- NSR=32, the network is in KNN mode

Distance represents the distance value between vector and the firing neuron with the closest model stored in its memory (i.e. top firing neuron). Distance is calculated by the neurons according to the Norm assigned to the neuron at the time it was committed (bit7 of the Context register). If no neuron fires, Distance=0xFFFF.

Category of the top firing neuron. It can range between 1 and 32767. Bit 15 is always set to 0 to mask the Degenerated flag. If no neuron fires, Category=0xFFFF.

NID is the identifier of the top firing neuron. NID can range between 1 and the number of neurons available in the network. If no neuron fires, Identifier=0.

Considerations before calling this function:

- Change the Global Context Register to match the context represented by the input vector
- Change the Network Status Register (NSR) to turn ON/OFF the KNN classifier, or to turn ON/OFF the Save-and-Restore mode.

Considerations after calling this function:

- Change the Network Status Register (NSR) to turn OFF the KNN classifier, if selected, prior to the next learning operation.
- If the NSR indicates a case of uncertainty (value 4 or 36), you can immediately execute a series of Read(DIST) + Read(CAT) to obtain the response of the next closest firing neurons, and this until you read a DIST=0xFFFF.

8.2 int RecognizeROI(int Left, int Top, int K, int *distances, int *categories, int *nids)

Recognizes the feature vector extracted from the ROI at the (Left, Top) location in the image and reports the distance, category and neuron identifier of the K closest neurons, if any.

The function returns the number of firing neurons or K whichever is the smallest. For example, if K=3, but only 2 neurons fire, the function returns, the value 2.

Distance represents the distance value between vector and the firing neuron with the closest model stored in its memory (i.e. top firing neuron). Distance is calculated by the neurons according to the Norm assigned to the neuron at the time it was committed (bit7 of the Context register). If no neuron fires, Distance=0xFFFF.

Category of the top firing neuron. It can range between 1 and 32767. Bit 15 is always set to 0 to mask the Degenerated flag. If no neuron fires, Category=0xFFFF.

NID is the identifier of the top firing neuron. NID can range between 1 and the number of neurons available in the network. If no neuron fires, Identifier=0.

Considerations before calling this function:

- Change the Global Context Register to match the context represented by the input vector
- Change the Network Status Register (NSR) to turn ON/OFF the KNN classifier, or to turn ON/OFF the Save-and-Restore mode.

Considerations after calling this function:

- Change the Network Status Register (NSR) to turn OFF the KNN classifier, if selected, prior to the next learning operation.

8.3 int FindROSOBJECTS(int stepX, int stepY, int skipX, int skipY, int *Xpos, int *Ypos, int* distance, int* category, int* nid)

Scan the Region of Search and report each recognized location.

The report consists of five output arrays Xpos, Ypos, distance, category, nid describing the recognized objects.

The scanning is made in a raster displacement with horizontal stepX and vertical stepY.

Xpos and Ypos are the center location of the region of interest

Distance, category and neuron identifier are the response of the 1st firing neuron, that is with the closest match.

The skipX and skipY options can accelerate the scanning process but must be used cautiously since positive recognition may be missed. If stepX is set to 1, a positive identification at a position X will set the next inspection at a position X + StepX. Similarly, if stepY is set to 1, any positive identification along a line Y will set the inspection of the next line of pixels at a position Y + StepY.

The output arrays must be sized to their maximum possible length which is $\frac{ROSW}{StepX} * \frac{ROSH}{stepY}$ and corresponds to a case where an object is recognized at each scanned position.

Considerations before calling this function:

- Verify that the ROI and FeatParams are the proper ones
- Verify that the ROS is the proper one
- Size the 5 output arrays [Xpos, Ypos, Distance, Category, Nid] to a length of $\frac{ROSW}{StepX} * \frac{ROSH}{stepY}$
- Change the Global Context Register to match the context represented by the input vector

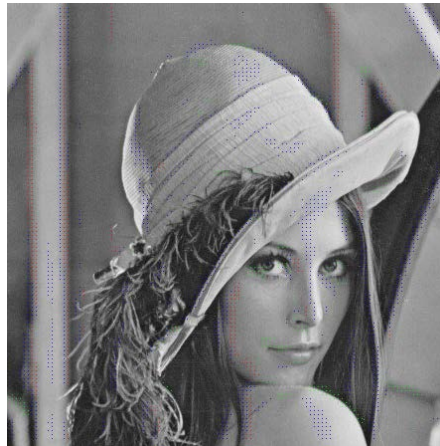
Example:

Map of Categories representing Vertical edges (red), Horizontal edges (green) and Diagonal edges (yellow)

ROS = 512 x 512

Step = 4 x 4

Recognized objects overlaid on the source image are spaced by stepY and stepY



8.4 int FindROSAnomalies(int stepX, int stepY, int MaxNbr, int *Xpos, int *Ypos)

Scan the Region of Search and report each location which is not recognized.

The scanning is made in a raster displacement with horizontal stepX and vertical stepY.

Xpos and Ypos are the center location of the region of interest

The output arrays must be sized to their maximum possible length which is $\frac{ROSW}{StepX} * \frac{ROSH}{stepY}$ and corresponds to a case where no object is recognized at each scanned position.

Considerations before calling this function:

- Verify that the ROI and FeatParams are the proper ones
- Verify that the ROS is the proper one
- Size the 5 output arrays [Xpos, Ypos, Distance, Category, Nid] to a length of $\frac{ROSW}{StepX} * \frac{ROSH}{stepY}$
- Change the Global Context Register to match the context represented by the input vector

8.5 int MapROS(int stepX, int stepY, int *CatMap, int *DistMap, int *NidMap)

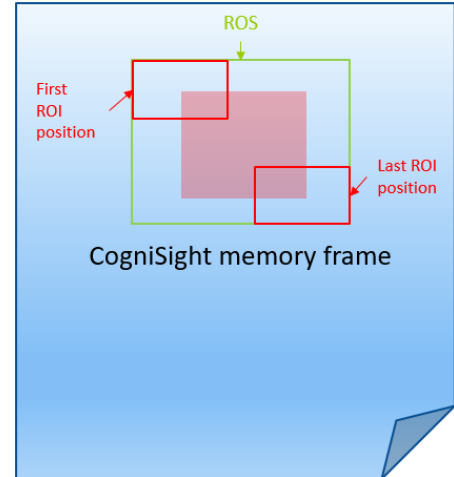
Builds the lists of the categories, distances and neurons' identifiers recognized at each step of a raster displacement within the Region of Search.

The three output have the same dimension L:

- L is the number of inspected positions of the ROI within the ROS
- L is a function of the selected StepX and StepY, and distance from the center of the ROI to the edges of the ROS
- These positions are within the shaded rectangle shown to the right
- $$L = \frac{ROS_W}{StepX} * \frac{ROS_H}{stepY}$$

The values of the CatMap array can range between [0,32,364].
 The values of the DistMap array can range between [0, 65535].
 The values of the NidMap array can range between [0, Number of neurons available].

The function returns MapWidth = $\frac{ROS_W}{stepX}$.



The 1D output arrays can be reshaped as 2D arrays. Their interpretation may require conversion into pseudo-color images (8-bit or higher) and displayed with appropriate color lookup table.

- The display of CatMap as 2D images show the spatial distribution of the categories in the image. If the application manages more than 256 categories of objects for the selected context, the values may require to be mapped down to a range of [0,255].
- DistMap reveals the distribution of the recognized areas per level of confidence. The color palette should associate a low distance value to a high confidence factor (like red color for high confidence) and vice and versa (like blue for a poor confidence). The distance values being much greater than 255, they should be scaled down using clipping, logarithmic or other interpolations.

Several color lookups (CLUT) are supplied in the Project folder.

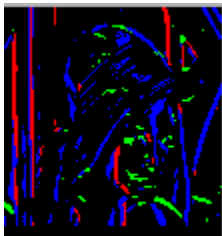
Example:

Map of Categories representing Vertical edges, Horizontal edges and Diagonal edges

ROS = 512 x 512

Step = 4 x 4

Map = 128 x 128



9 KNOWLEDGE LEVEL FUNCTIONS

9.1 int SaveProject(char *filename)

Saves to file the knowledge stored in the neurons as well as the current ROI size, the feature extraction and its parameters and the settings of the neural network (maxif, minif);

The function returns the number of saved neurons (ncount) after the upload to the NeuroMem chip(s).

The format of the file is composed of a header followed by the neurondata or an array describing the content of the “neurons”:

Header information

- ROI size
- Feature ID
- Feature Parameters
- ROS coordinates

NeuronData is an array of (neuronSize + 4) integers as follows:

- NeuronData[0]= NCR, Neuron Context Register
- NeuronData[1,NeuronSize] = neuron’s memory or NeuroSize components
- NeuronData[NeuronSize+1]= AIF, Active Influence Field
- NeuronData[NeuronSize+2]= MINIF, Minimum Influence Field
- NeuronData[NeuronSize+3]= CAT, Category

9.2 int LoadProject(char *filename)

Restores from file the content of the neurons as well as the current ROI size, the feature extraction and its parameters and the settings of the neural network (maxif, minif);

The function returns the actual number of committed neurons (ncount) after the upload to the NeuroMem chip(s).

The format of the file is composed of a header followed by the neurondata or an array describing the content of the “neurons”:

Header information

- ROI size
- Feature ID
- Feature Parameters
- ROS coordinates

NeuronData is an array of (neuronSize + 4) integers as follows:

- NeuronData[0]= NCR, Neuron Context Register
- NeuronData[1,NeuronSize] = neuron’s memory or NeuroSize components
- NeuronData[NeuronSize+1]= AIF, Active Influence Field
- NeuronData[NeuronSize+2]= MINIF, Minimum Influence Field
- NeuronData[NeuronSize+3]= CAT, Category

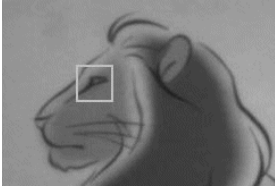
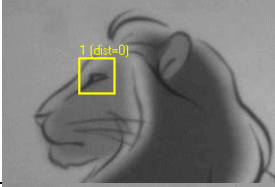
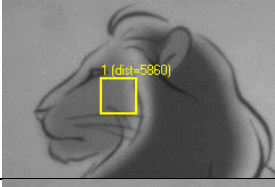
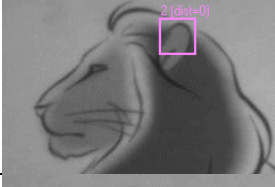
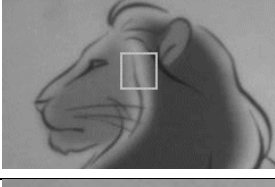
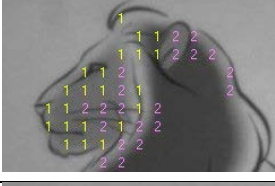

The CogniSight SDK is developed on top of the CogniPat SDK which includes additional functions to control the neurons and manipulate data-agnostic vectors. For more information, refer to CogniPat.h header file and the [CogniPat SDK manual](#).


Among the most commonly used functions:

- AvailableNeurons
- CommittedNeurons
- ClearNeurons

10 APPENDIX B: TUTORIAL, HINTS AND TIPS

10.1 Supervised Object Learning

| Description | | Instructions | Comment |
|--|---|--|---|
| Size and position the ROI to include the eye of the lion and learn as category 1 |  | SizeROI MoveROI LearnROI (1) Read Neurons=1 | A first neuron is committed. It holds the model of the lion's eye and its associated category #1. |
| Recognize the ROI without moving it |  | RecognizeROI Read Distance =0 Read Category=1 | The Distance 0 indicates that the signature of the ROI matches exactly the model of a neuron associated to category 1. |
| Move the ROI by a few pixels and recognize |  | MoveROI RecognizeROI Read Distance =5860 Read Category=1 | The increase of the Distance value indicates that there is a drift between the signature of the ROI and the model of the closest (and only) neuron. |
| Move the ROI over the ear of the lion and learn as category 2. |  | MoveROI LearnROI (2) Read Neurons=2 | A 2nd neuron is committed. It holds the model of the lion's ear and its associated category #2 |
| Move the ROI in between the eye and the ear of the lion. |  | RecognizeROI Read Distance =0xFFFF | The Distance 0xFFFF indicates that the signature of the ROI is not recognized by either neuron which is the expected response. |
| Overview of the recognized objects |  | Set ROS SearchROS Read the VObjects and tag their position and category in the image | The 1s and 2s show respectively the locations recognized as the lion's eye and ear. The two neurons are over-generalizing and must be corrected. |
| Move the ROI over the nose of the lion and learn as background or category 0 |  | MoveROI LearnROI(0) Read Neurons=2 | No new neuron is committed but neurons 1 and 2 shrink their influence fields to stop recognizing the region as the eye or ear of the lion. |

| | | | |
|------------------------------------|---|--|--|
| Overview of the recognized objects |  | Set ROS SearchROS Read the VObjects and tag their position and category in the image | The accuracy of the recognition is now satisfactory. |
|------------------------------------|---|--|--|

10.2 Using multiple feature extractions for object classification

The following example illustrates how to learn and recognize objects based on two different features for more robustness.

Let's take the example of character recognition.



Combining the use of a subsample vector and a histogram vector can help discriminate certain hand written digits.

In this case, two sub-networks of neurons will be trained to recognize the same input objects based on 2 feature vectors. The subsample can be assigned to the context 1 and the histogram to the context 2. The change of context must occur prior to the functions LearnROI, RecognizeROI and FindROSOBJECT, MapROS. The change of feature is executed by the SetFeatParams function.

Example1: Learn a same example using 2 feature vectors

```
NeuroMem.GCR=1
SetFeatParams(FeatID1, norm1, minif1, maxif1, FeatParam11, FeatParam12, FeatParam13, FeatParam14)
LearnROI(X,Y,Cat1)
NeuroMem.GCR=2
SetFeatParams(FeatID2, norm2, minif2, maxif2, FeatParam21, FeatParam22, FeatParam23, FeatParam24)
LearnROI(X,Y,Cat1)
```

Example2: Recognize a same example using 2 feature vectors

```
NeuroMem.GCR=1
SetFeatParams(FeatID1, norm1, minif1, maxif1, FeatParam11, FeatParam12, FeatParam13, FeatParam14)
[Cat1out, Dist1out]=RecoROI(X,Y)
NeuroMem.GCR=2
SetFeatParams(FeatID2, norm2, minif2, maxif2, FeatParam21, FeatParam22, FeatParam23, FeatParam24)
[Cat2out, Dist2out]=RecoROI(X,Y)
If (Cat1out==Cat2out) printf("double score!")
```

10.3 What if an object appears at different scale factors?

The neurons assigned to a given context C and trained to recognize objects with a size W x H will be able to recognize the same object at different scales provided that the ratio of its primitive blocks remains the same.

If an object is taught using the following settings:


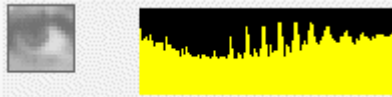


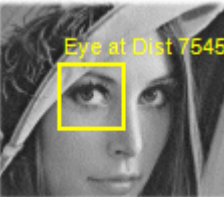

ROI size=rS

Block size=bS

The same neuron will recognize the object viewed at a scale N with the following settings:

ROI size= N x rS

Block size= N x bS

| | | |
|---|---|--|
|  | <p>This original image is used to teach an example of an eye.</p> | <p>ROI 32x32 Block 2x2</p>  |
|  | <p>Case 1: Image is zoomed out. The ROI and block size are both reduced using the same ratio of 1/2</p> <ul style="list-style-type: none"> ⇒ The feature vector is similar to the one extracted in the original image. ⇒ The neuron trained on the original image has a good chance to recognize this vector. | <p>ROI 16x16 Block 1x1</p>  |
|  | <p>Case2: Image is zoomed out. The ROI and block size are kept the same</p> <ul style="list-style-type: none"> ⇒ The feature vector encodes different information than the one extracted in the original image. ⇒ The neuron trained on the original image has less chance to recognize this vector. | <p>ROI 32x32 Block 2x2</p>  |

10.4 Dealing with objects of different sizes

The following example illustrates how to learn and recognize different parts of an object.

ROI #1 = Filling level of a bottle
Size = 64 x 8
FeatID = Vertical profile
Category= acceptable, too_low, too_high
Will be assigned to Context 1

ROI#2 = Front label must be good
Size = 128 x 128
FeatID = SubsampleRGB
Category= acceptable, slanted, scratched, folded
Will be assigned to Context 2



The change of context must occur prior to the broadcast of the corresponding type of feature vector.

10.4.1 Example 1: Learn two filling levels sampled at a same location in two consecutive images

```
SetROI(Width1, Height1)
SetContext(1, 2, 0x4000)
GrabImage()
LearnROI(Left, Top, acceptable)
GrabImage()
LearnROI(Left, Top, too_low)
```

10.4.2 Example 2: Learn two labels sampled at different X locations in a same image

```
SetROI(Width2, Height2)
SetContext(2, 2, 0x4000)
LearnROI(Left1, Top, acceptable)
LearnROI(Left2, Top, acceptable)
```

10.4.3 Example3: Inspect the two ROIs and accept the bottle if both label and filling level are correct

```
SizeROI(Width1, Height1)
SetContext(1, 2, 0x4000)
MoveROI(Left1, Top1)
RecoROI((Left1, Top1, out Distance1, out Category1)
If (Category1 != acceptable) return("Fail");
SizeROI(Width2, Height2)
MoveROI(Left2, Top2)
SetContext(2, 2, 0x4000)
RecoROI(Left2, Top2, out Distance2, out Category2)
f (Category2 != acceptable) return("Fail");
else return("Pass")
```

10.5 Surface Inspection

Texture learning is easy with the CogniSight engine. A region of interest can be divided into patches and the neurons will automatically learn the patches which are significant to describe the texture of the region.

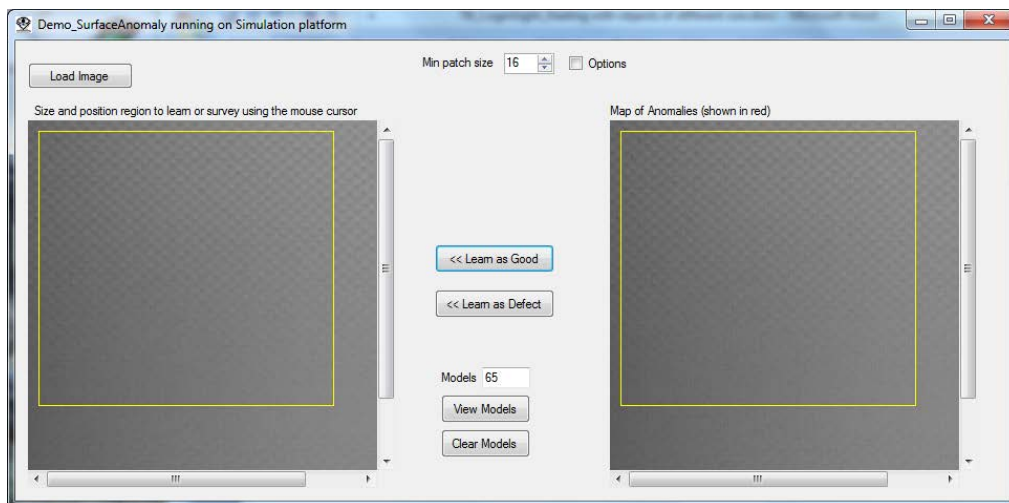
In the example below, the surface inspected is solar glass which features a periodic bumpy pattern. Following is a series of patches of 16x16 pixels learned by the neurons. Assuming that a glass area with good quality is learned by taking examples of patches at all possible phase and assigning them the “Good” category, the content of the resulting committed neurons is a description of the good glass texture.

Example patches of 16x16 pixels

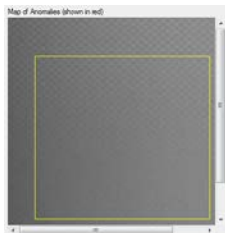


The user interface presented below is very simplistic but sufficient to illustrate how to develop a surface inspection system with the CogniSight technology. The area selected by the user and outlined in yellow has been learned as a “Good” texture and this has generated 65 models. The number of models depends on two settings of the learning operation: the value of the maximum influence field (MAXIF) of the neurons and the scanning step used to extract the sample patches from the region of interest.

- The higher the step and the smaller the number of samples.
- The smaller the MAXIF, the more models.

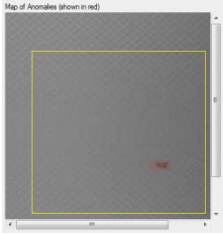


The image to the right is supposed to highlight the patches which are not recognized by the CogniSight engine because they do not match any of the 65 models. In this case, all learned patches are positively identified.



The same remark is true if the region of interest is moved around as shown in the image to the left.

This is made possible by learning the content of the region using a step of 1 or 2 which allows to generate representations of the patches of texture at many different phases:



If a new image is loaded and shows a significant defect, the neurons will not recognize the patches at the location of the defect. They appear highlighted in red in the Transform image.

In the event that a defect is not properly identified, a new region limited to patches covering the defect can be selected with the mouse cursor and learned as a Bad texture. This learning operation will have the effect to reduce the influence field of the neuron(s) recognizing the patches as good prior to learning them as counter examples.

More information...

General Vision has developed a complete [Defect Detection System](#) installed and tested over a glass float. It is revolutionary solution based on a scalable chain of V1KU cameras programmed with a same CogniSight engine for surface inspection and loaded with the same knowledge in their respective CM1K chips.