CogniSight SDK
User's Manual

IMAGE RECOGNITION BASED ON NEUROMEM NEURAL NETWORKS

Version 5.4.2
Revised 07/01/2019

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

CONTENTS

1 INTRODUCTION

The CogniSight SDK is a software development kit for image learning and recognition powered by a single or multiple NeuroMem neural networks. 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.

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

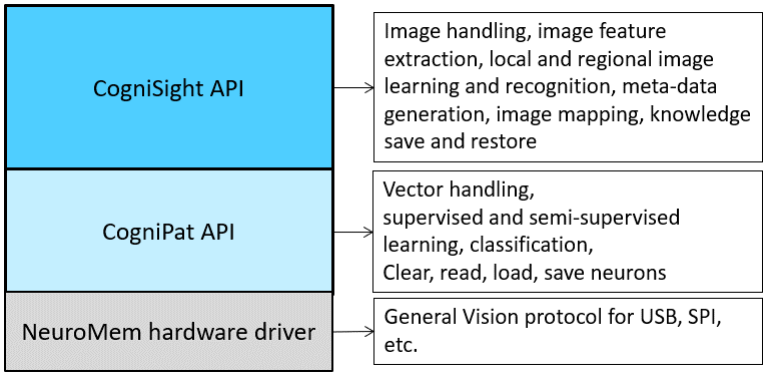
1.1 The NeuroMem neural network

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.

1.2 CogniPat library

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 image learning, classification, knowledge saving and restore. Several versions of the DLL exist with hardware specific drivers necessary to communicate with the selected platform:

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

All DLLs have the same entry points, so they are fully interchangeable. This means that an application developed with the CogniSight SDK can run on all the compatible platforms by simply changing the DLL linked to the application.

The selection of the CogniSight DLL is made as follows depending on the programming environment:

	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

2.2 Files and Folders

- Bin folder
  - 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
- Examples folder
  - Examples are provided to helps understand how to use the neurons to learn and recognize visual objects or events.

2.3 Image Knowledge Builder software

The CogniSight SDK is delivered with a standard version of the Image Knowledge Builder (IKB) so you can experiment with a variety of learning and recognition methods on your own images prior to programming your application with the SDK.

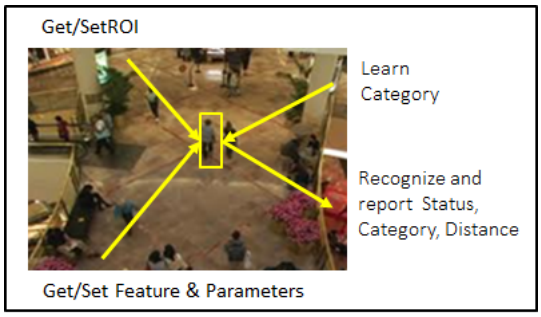
IKB is entirely developed with the CogniSight library. The current version lets you use the neurons to build a recognition engine based on single type of feature extraction at a time. However, its user interface lets you easily evaluate and compare different features and settings while reusing the same training sets (images and annotations).

3 LIBRARY CONTENT

The CogniSight library manipulated the following data types:

- Images (input) and Maps (output)
- Region of Interest or ROI
- Region of Search or ROS

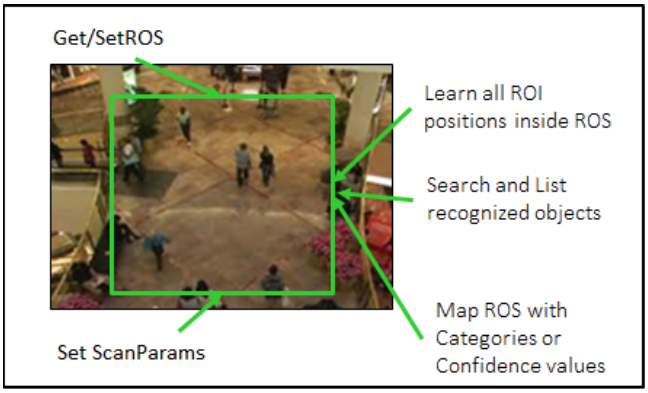
A Region Of Interest (ROI) is the primitive area to learn or recognize. It can be a discrete object, part of an object, a significant feature in a scene, a patch of texture, etc.



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 ROI Properties and Methods are described chapter 6.

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 step xy and type of displacement.



The ROS properties and Methods are described Chapter 7.

4 HARDWARE INTERFACE

4.1 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=NeuroMem hardware associated to the CogniSight_xyz.dll linked to your application.	Reserved. Default is 0.

The selection of the CogniSight_xyz.dll is made as follows depending on the programming environment:

	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

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.

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

Read the specifications of the silicon neurons:
- neuronSize, memory capacity of each neuron in byte
- neuronsAvalaible, number of neurons available

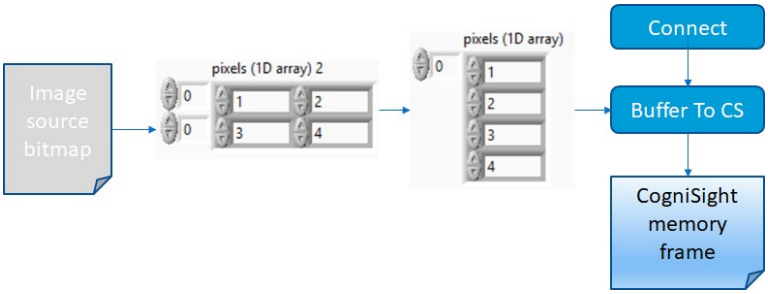
The number of neurons available is returned by the Connect function in the case of a platform with fixed capacity, and CountNeuronsandRset in the case of a platform with variable capacity.

4.3 int Disconnect();

Closes the communication with the current device.

5 IMAGE MANAGEMENT

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.



At the time of the transfer, you must define the dimensions of the CogniSight memory frame: Width, Height and Byte-Per-Pixel (bpp). In the case of a monochrome plane, each byte represents the grey level intensity of a pixel. In the case of a color plane three consecutive bytes represent the succession of R, G, B intensities of a pixel.

- The CogniSight memory frame can be used to store the entire source image or only a region of the source image. In the latter case, you will simply have to remember that recognized locations reported by the CogniSight engine will have the upper-left corner of this region as point of origin.

- The CogniSight memory frame can be used to store the RGB information of the pixels, if applicable, or only their grey level information. Indeed if it is known that the neurons are to be trained on a monochrome feature, this will optimize memory allocation and processing speed without preventing you to display the original color image in your graphical interfaces.

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

Load a pixel array into a memory frame of the CogniSight workspace. 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 byte array represents a color image and BytesPerPixels value is 3, the function expects an imageBuffer with interlaced red, green, blue pixel values.

imageBuffer is a byte array with length equal to the product (Width * Height * Bytes Per Pixel).

If the array represents a monochrome image or portion of a monochrome image, it is a series:
Pixel[0], Pixel[1], …Pixel[Width], Pixel[Width+1],…Pixel[Width * Height].

If the array represents a color image or portion of a color image, it is a series:
PixelR[0], PixelG[0], PixelB[0], PixelR[1], PixelG[0], PixelB[0], …PixelR[Width * Height], PixelG[Width * Height], PixelB[Width * Height].

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

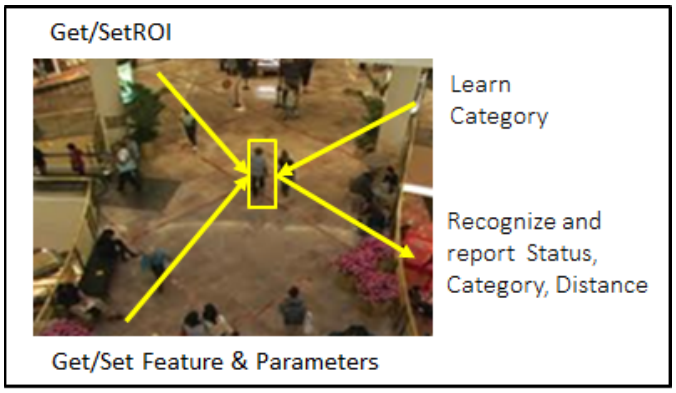
Reads the parameters Width, Height and BytePerPixel describing the current CogniSight memory frame.

5.3 void CSToBuffer(unsigned char *imageBuffer)

Read the CogniSight memory frame and returns it as a byte array. The byte array must be sized to at least Width * Height * BytePerPixel. These three values can be retrieved using the function CSBuffInfo parameters. This function can be useful to retrieve an image which was directly acquired on a compatible hardware with on-board sensor.

6 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.



6.1 void Get/SetROI(int Width, int Height)

Read or Define the nominal width and height of the current ROI.

6.2 void Get/SetFeat(int FeatID)

Read or Define the feature extraction method to apply to the current ROI.

The CogniSight SDK offers a set of common feature extractions listed below. They are simple but convey good discrimination capabilities when used in conjunction with the powerful NeuroMem RBF classifier. Custom features can also be extracted from an ROI, as described in a next chapter.

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

The selection of the feature extraction method does not have to be necessary a “color” feature because an image is a 24-bit color image. It must rather be a discriminating characteristic for the targeted objects and application.

For example, if your application consists of verifying the alignment of an object by detecting that its edges, the color information might be irrelevant and a grey level subsample sufficient to detect aligned and misaligned patterns.

Discarding the color information will eliminate unnecessary variability and allow the calculation of a very simple feature with less data compression.

6.3 void Get/SetFeatParams(int FeatID, int Normalize, int Minif, int Maxif, int Param1, int Param2)

Each feature extraction method (FeatID) uses 3 mandatory parameters and two optional parameters:

- 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 2nd 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.	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

(*) Refer to the Chapter Feature Extractions for more details.

6.4 void SizeSubsample(int Width, int Height, int Monochrome, int KeepRatio);

Calculates Param1 and Param2 for the extraction of the subsample, whether monochrome (featID=0) or color (featID=3). These 2 parameters are the width and heights of the 256 square blocks fitting inside the ROI. Their values can be read using the GetFeatParams function.

(*) Refer to the Chapter Feature Extractions for more details.

6.5 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.

6.6 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 commited 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).
    - 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 category 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.
    - The Save and Restore mode is not compatible with the learning mode.

6.7 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 agree with its category (IDentified)
- NSR=4, the vector is recognized but the firing neurons disagree 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.

6.8 int ClassifyROI(int Left, int Top, int K, int *distances, int *categories, int *nids)

Classifies 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.

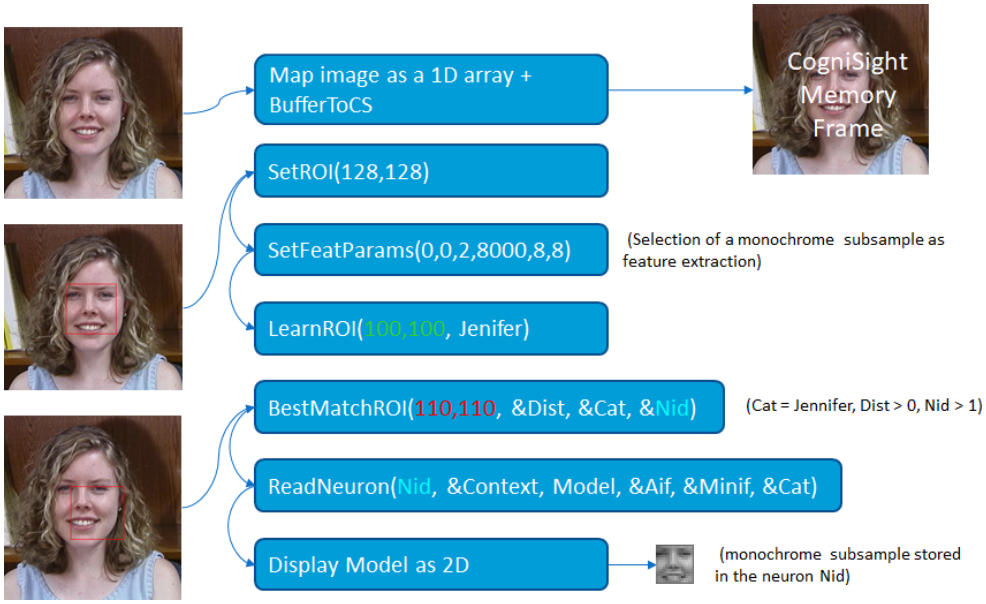
Considerations before calling this function:

- 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.

6.9 Example of a single ROI manipulations



6.10 Multiple ROIs manipulations

If an application must inspect different ROIs per part, it shall store the definition of the different ROIs and make sure to change the Global Context of the neurons when switching between ROIs. IN addition to the simple examples below, please refer to the last Chapter of this manual: Building a Multiple Expert system.

6.10.1 Multiple ROIs, Multiple features

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

```
SetROI(Width1, Height1)
SetFeatParams(FeatID1, Norm1, Minif1, Maxif1, Param11, Param12);
SetContext(1, 2, 0x4000)
MoveROI(Left1, Top1)
RecoROI((Left1, Top1, out Distance1, out Category1)
If (Category1 != acceptable) return("Fail");
```

```
SetROI(Width2, Height2)
SetFeatParams(FeatID2, Norm2, Minif2, Maxif2, Param21, Param22);
MoveROI(Left2, Top2)
SetContext(2, 2, 0x4000)
RecoROI(Left2, Top2, out Distance2, out Category2)
f (Category2 != acceptable) return("Fail");
else retun("Pass")
```

6.10.2 Same ROI, Multiple features

The classification of the ROI can be based on 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.

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!")
```

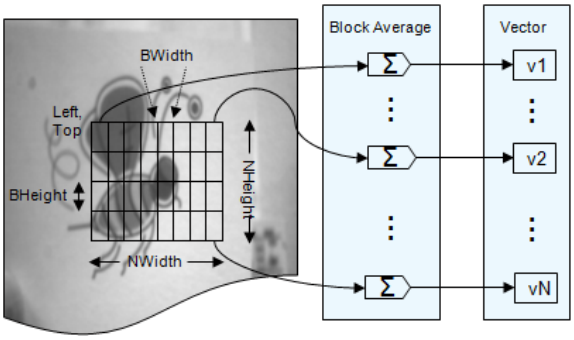
7 FEATURE EXTRACTIONS

SubSample

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.

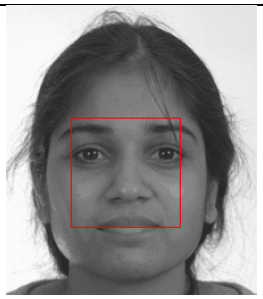
The Subsample and SubSampleRGB functions take to input parameters which describe the size of the internal blocks to average, BWidth and BHeight as described below.

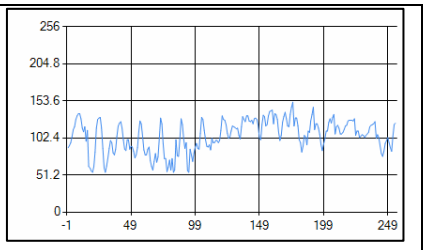
- The pixels of block #i are averaged to produce the i^{th} component of the signature vector.
- The relationship between the four parameters is :
  - NWIDTH= n*BlockWIDTH
  - NHEIGHT= m*BlockHEIGHT
  - n*m <=256 in the case of a monochrome subsample
  - n*m <=85 in the case of an R+G+B subsample



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

Since the SubSample function generates 1 average value per block and the final vector can only have 256 values, the blocks must be sized such that their number inside the ROI is less than or equal to 256.

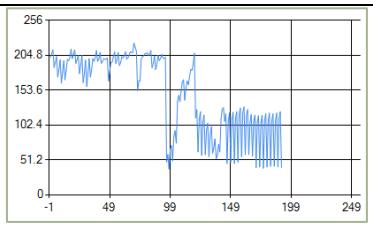




Feature vector reconstructed as a monochrome image

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. The ROI must contain less than 85 blocks so 3 x 85 values fits in the 256 bytes of memory of the neurons.

Since the SubSampleRGB function generates 3 average value per block and the final vector can only have 256 values, the blocks must be sized such that their number inside the ROI is less than or equal to 85.

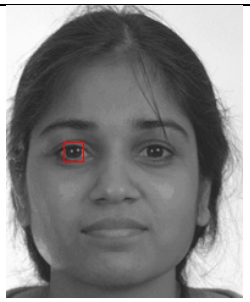


Feature vector reconstructed as an RGB image

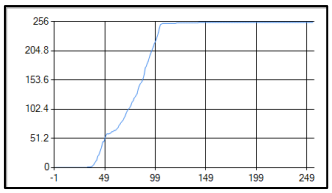
7.1 Histograms

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.
The amplitude of the histogram is scaled down to a byte value by multiplying it by the ratio (255/N) with N, the number of pixels in the ROI.

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.



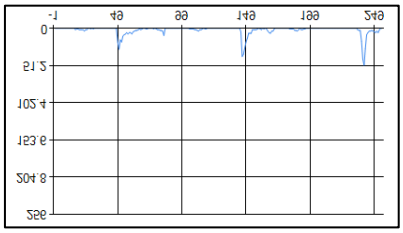
Cumulative histogram



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.

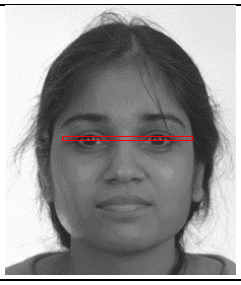
HistoCumulRGB or cumulative color histogram is a mapping of the standard histogram which counts the cumulative number of pixels in all the bins up to the specific bin.

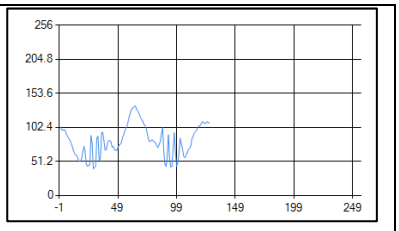
RGB Histogram



7.2 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.





7.3 Custom feature extraction

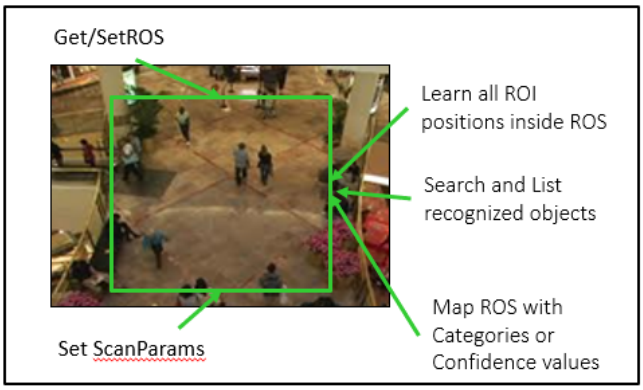
If you wish to develop a feature extraction method which is not provided in the CogniSight SDK, it can be grafted in your application using the functions of the CogniPat SDK(*) as described in the table below:

	Built-in feature extraction	**Custom feature extraction**
Pixel data	Image source must be transferred to the CS image plane with the function BufferToCS	Bitmap of the image source , or the same buffer as the one used by the function BufferToCS
Feature extraction	SetFeat SetFeatParams	CustomFeat (X, Y), your function manipulating the pixel data FeatVector, your output vector formatted as a byte array of maximum length 256
Learn locally	LearnROI (X, Y)	FeatVector= CustomFeat(X, Y) Learn(FeatVector)
Recognize locally	ClassifyROI (X,Y)	FeatVector= CustomFeat(X, Y) Classify(FeatVector)
Learn an area	SetROS LearnROS ()	For each X,Y over the ROS area FeatVector= CustomFeat(X, Y) Learn(FeatVector)
Recognize an area	SetROS FindROSObjects ()	For each X,Y over the ROS area FeatVector= CustomFeat(X, Y) Classify (FeatVector)

(*) 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.

8 REGIONS 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 step xy and type of displacement.

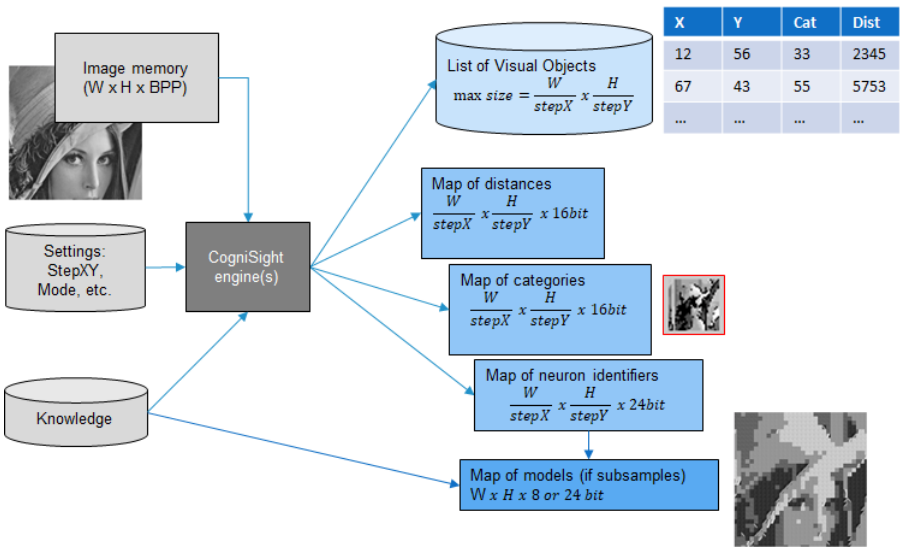


Learning an ROS can involve supervised and semi-supervised functions.

Surveying a Region of Search can produce multiple types of outputs.

- 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.

- A Map **(MAP)** is a transform image of a Region of Scan reporting the spatial distribution of the recognized objects in term of category value, distance/confidence value and neuron identifiers. The dimension of the map is the dimension of the ROS divided by the selected scanning step.



Note that if the CogniSight memory frame only represents a portion [Left, Top, Right, Bottom] of the original image, the recognized locations reported by the Find and Map functions have the position [Left, Top] as point of origin. You will have to translate them back to the referential of the source image if applicable to your GUI.

8.1 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.

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

Reads the current Region Of Search

8.3 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.

8.4 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.

8.5 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 recognized as novelty by the neurons are automatically learned and assigned a category value as defined by the CatAllocMode parameter.

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

- 0: constant value
- 1: auto-increment by 1

- 2: maximum delta between the vectLen components
- 3: average value of the vectLen components
- 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.

For an example on how to build and use a codebook, refer to the Appendix Surface Inspection.

8.6 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 recognized regions 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 positon 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 positon 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

8.7 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.

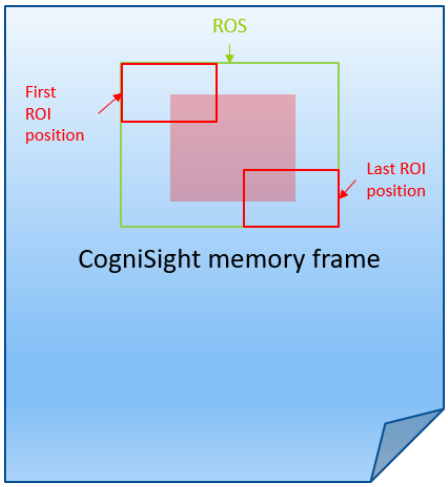
Considerations before calling this function:
- Verify that the ROI and FeatParams are the proper ones

- 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

8.8 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= = $\frac{ROSW}{StepX} * \frac{ROSH}{stepY}$



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 or the number of samples representing the width of the transform image. This value is used to reconstruct the Transform image.

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.

9 COGNISIGHT PROJECT FILES

A CogniSight 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.

The CogniSight Project format described below is compatible with the Image Knowledge Builder and all the CogniSight libraries and CogniSight based applications.

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

9.3 Header details

The information of the header is easily retrieved with the GetROI, Get FeatParams and GetROS functions. However if you want to retrieve the information from the binary file, the header is a series of 4 bytes formatted as [Module, Register, Data_UpperByte, Data_LowerByte]. The end of the header is marked by four consecutive bytes set to 0xFF, after which you can access the neuron data.

```
#define MOD_TOP             0x52    #define GV_VERSION          0x62

#define MOD_NM              0x01    #define NM_LEFT             0x11
                                    #define NM_TOP              0x12
                                    #define NM_NWIDTH           0x13
                                    #define NM_NHEIGHT          0x14
                                    #define NM_BWIDTH           0x15
                                    #define NM_BHEIGHT          0x16
                                    #define NM_RSR              0x1C
                                    #define NM_RTDIST           0x1D
                                    #define NM_RTCAT            0x1E
                                    #define NM_ROIINIT          0x1F

#define MOD_CS              0x10    #define CS_WIDTH            0x81
                                    #define CS_HEIGHT           0x82
                                    #define CS_FEATID           0x83
                                    #define CS_FEATNORMALIZE    0x84
                                    #define CS_FEATMINIF        0x85
                                    #define CS_FEATMAXIF        0x86
                                    #define CS_FEATPARAM1       0x87
                                    #define CS_FEATPARAM2       0x88
                                    #define CS_ROSLEFT          0x89
                                    #define CS_ROSTOP           0x8A
                                    #define CS_ROSWIDTH         0x8B
                                    #define CS_ROSHEIGHT        0x8C
                                    #define CS_STEPX            0x8D
                                    #define CS_STEPY            0x8E
                                    #define CS_SKIPX            0x8F
                                    #define CS_SKIPY            0x90
```

10 BUILDING A MULTIPLE EXPERTS SYSTEM

The CogniSight SDK lets you design applications based on multiple experts trained on identical or different ROIs using identical or different features. These experts can be used to make combinatorial or hierarchical decisions. For example, an application with a high cost of mistake may require that at least N experts produce a same classification in order consider the overall response as a positive identification.

The present version of the CogniSight SDK does not manage the definition of multiple expert, so you will have to save the description of your experts as illustrated below in a file with the format of your choice. The next version will have this built-in the library.

Example with two experts

Expert #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

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

10.1 Assigning a context value to an expert

A context value should be considered as an index to a list of experts.
You can have multiple experts trained on the same images and annotations but using different feature extractions
- Context 1: subsample 32x64 with 2X4 blocks
- Context 2: subsample 32x64 with block 6x32
- Context 3: HistogramRGB

You can have multiple experts trained using a same feature but on different types of images
- Context Zoom 1, Context Zoom 5, Context Zoom 10
- Context Daytime and Context Nighttime

The context is a user-defined index value tying together the nominal size of the ROI, a type of feature, a scale, etc.

The context must be set to the correct value prior to broadcasting a vector. This operation is only necessary if the vector represents a new dimension (aka a different feature, or ROI size, or image scale, etc.)

The context of a neuron is "frozen" at the time it gets committed. This is equivalent to assigning its index to a list of experts in use.

10.2 Using Context 0

The context 0 is a special value in the sense that it activates all the neurons regardless of their individual context value.

The context 0 should not be used during learning, except if your application uses only one expert.

10.2.1 Example #1

You train 3 experts called Context Zoom 1, Context Zoom 5, Context Zoom 10, and using the same feature extraction. At the time of the recognition, you select Context 0 and build a consensus if at the maximum 2 of the experts agree with the recognized category. For example, you would consider a response as false positive if Context Zoom 1 and Context Zoom 10 agree, but not Context Zoom 5. The fact that one scale is skipped could be suspicious

10.2.2 Example #2

You train 2 experts Context Daytime and Context Nighttime to recognize a target but activate them both at dawn and dusk.

10.3 Saving projects based on multiple experts

The present version of the CogniSight SDK does not manage the definition of multiple expert, so you will have to save the description of your experts as illustrated below in a file with the format of your choice. The next version will have this built-in the library.

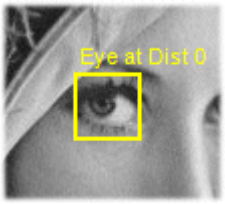
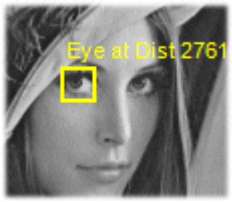
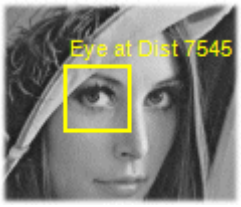
The SaveProject and LoadProject functions save and restore the knowledge of all the experts at once, but only the definition of the active expert at the time of the save. So, if your application manages multiple experts, it is mandatory that you save their description and their associated context values in a separate text file.

11 APPENDIX B: TUTORIAL, HINTS AND TIPS

11.1 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

	This original image is used to teach an example of an eye.	ROI 32x32 Block 2x2 
	Case 1: Image is zoomed out. The ROI and block size are both reduced using the same ratio of ½ ⇨ 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.	ROI 16x16 Block 1x1 
	Case2: Image is zoomed out. The ROI and block size are kept the same ⇨ 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.	ROI 32x32 Block 2x2 

11.2 Surface Inspection

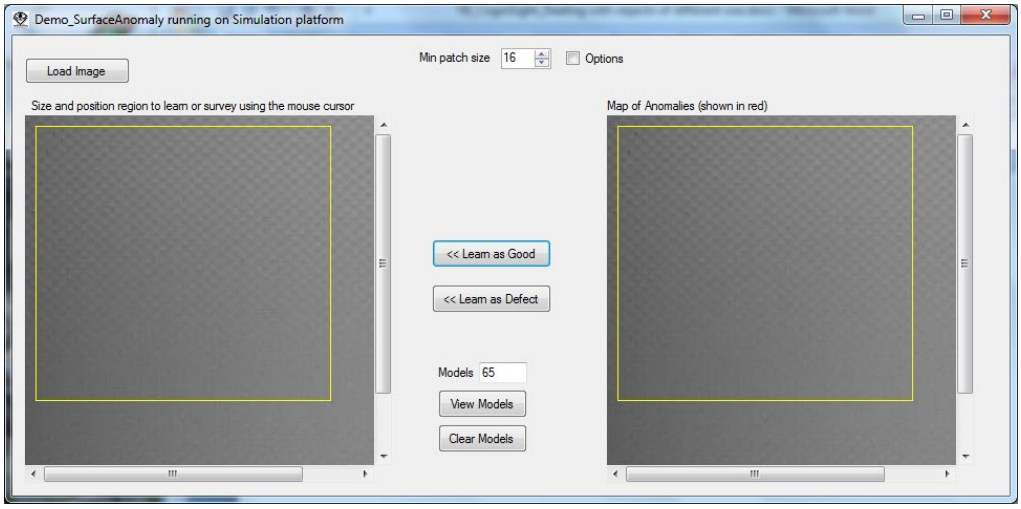
Texture learning is easy with the CogniSight engine. A region of interest can be divided into patches and the neurons will automatically learn the patches which are significant to describe the texture of the region.

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. If 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.

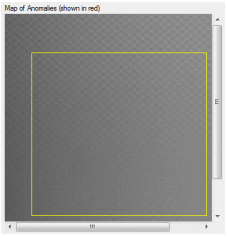
Example patches of 16x16 pixels 

The user interface presented below is very simplistic but enough 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.

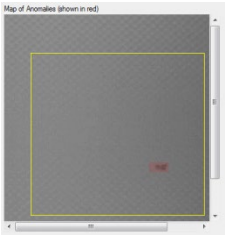


The image to the right is supposed to highlight the patches which are not recognized by the CogniSight engine because they do not match any of the 65 models. In this case, all learned patches are positively identified.



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.

If 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.