CogniSight SDK
User's Manual

SOFTWARE DEVELOPMENT KIT FOR IMAGE RECOGNITION BASED ON NEUROMEM SILICON NETWORK

Version 4.3.1
Revised 01/15/2018

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

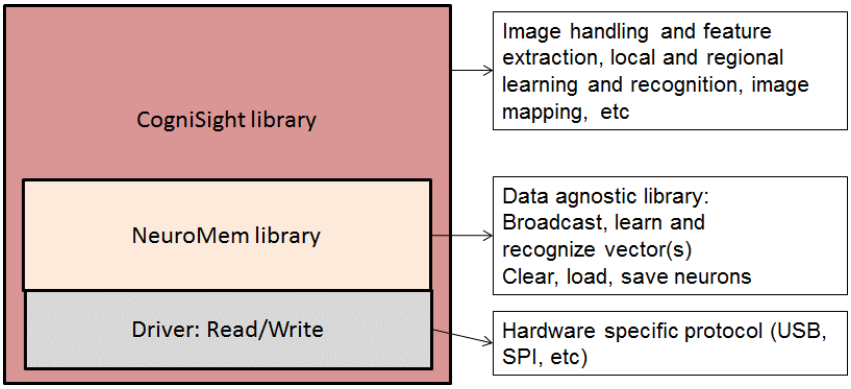
CONTENTS

Introduction 4
Package Content 5
Bin Folder 5
Examples 5
Concepts and definitions 6
Image Frame 6
Region of Interest 6
Region of Scan 6
Visual Objects 7
Maps 7
Knowledge 7
Project 7
Feature Extraction 7
Multiple Contexts 8
Deep Learning 8
Dichotomy of Possible applications 9
Hardware Interface Functions 10
int Version(); 10
int Connect(int Platform, int DeviceID); 10
int Disconnect(); 10
void getNeuronsInfo(int* neuronSize, int* neuronsAvailable) 10
int CountNeuronsandReset(); 10
Image management 11
void BufferToCS(unsigned char *imageBuffer, int Width, int Height, int BytesPerPixel) 11
void GetCSBuffInfo(int *Width, int *Height, int *BytePerPixel) 11
void CSToBuffer(unsigned char *imageBuffer) 11
Region of INterest (ROI) 12
void SetROI(int Width, int Height) 12
void GetROI(int *Width, int *Height) 12
void SetFeat(int FeatID) 12
void GetFeat(int *FeatID) 12
void SetFeatParams(int FeatID, int Normalize, int Minif, int Maxif, int Param1, int Param2) 12
void GetFeatParams(int *FeatID, int *Normalize, int *Minif, int *Maxif, int *Param1, int *Param2) 13
Int(length) GetFeature(int Left, int Top, int *Vector) 13
void SizeSubsample(int Width, int Height, int Monochrome, int KeepRatio); 13
REgions of Search (ROS) 14
void SetROS(int Left, int Top, int Width, int Height) 14
void GetROS(int *Left, int *Top, int *Width, int *Height) 14
Int(vectorNbr) GetROSVectors(int stepX, int stepY, unsigned char *Vectors, int *VLength) 14

Learning functions 15
int LearnROI(int Left, int Top, int Category) 15
int LearnROS(int stepX, int stepY, int category) 15
int BuildROSCodebook(int stepX, int stepY, int CatAllocMode) 15
int ROSToNeurons(int stepX, int stepY, int UsePositionsAsContext) 16
Recognition functions 16
Int(nsr) BestMatchROI(int Left, int Top, int *distance, int *category, int *nid) 16
int RecognizeROI(int Left, int Top, int K, int *distances, int *categories, int *nids) 17
int FindROSObjects(int stepX, int stepY, int skipX, int skipY, int *Xpos, int *Ypos, int* distance, int* category, int* nid) 17
int FindROSAnomalies(int stepX, int stepY, int MaxNbr, int *Xpos, int *Ypos) 17
FindROS Saliencies 18
void FindROSSalientBlocks(int stepX, int stepY,int K, int *CtrX, int *CtrY, int *AIF) 19
void FindROSUniqueBlocks(int stepX, int stepY,int K, int *CtrX, int *CtrY) 19
Transform Functions 20
int MapROS(int stepX, int stepY, int *CatMap, int *DistMap, int *NidMap) 20
Knowledge Level Functions 21
int SaveProject(char *filename) 21
int LoadProject(char *filename) 21
Appendix A: Feature Extractions 23
SubSample 23
Histogram 23
Histogram Cumulative 23
SubsampleRGB 23
Histogram RGB 24
Histogram RGB Cumulative 24
Horizontal, Vertical or Composite Profiles 24
Appendix B: Tutorial, Hints and Tips 25
Supervised Object Learning 25
Using multiple feature extractions for object classification 26
What if an object appears at different scale factors? 27
Dealing with objects of different sizes 28
Example 1: Learn two filling levels sampled at a same location in two consecutive images 28
Example 2: Learn two labels sampled at different X locations in a same image 28
Example3: Inspect the two ROIs and accept the bottle if both label and filling level are correct 28
Surface Inspection 29

INTRODUCTION

The CogniSight SDK is a software development kit for image learning and recognition based on a NeuroMem neural network.



The CogniSight SDK is developed on top of the CogniPat SDK and therefore includes all the entry functions of the CogniPat SDK which are agnostics to data types. They can be used to broadcast custom feature vectors extracted from images, but also waveforms, text, etc. These functions are described in the CogniPat SDK manual.

The CogniSight library lets you teach objects or textures from reference images. Search functions can report the list of recognized objects with their category and a confidence factor, or the list of anomalies or novelties. Transform functions can produce distribution maps per category and confidence. To increase the accuracy of a recognition engine, you can train multiple sub-networks associated to different sensor inputs and/or different feature extractions. The final recognition can then build a more robust decision by consolidating the response of these multiple networks and weighting cases of unknown and uncertainty.

The CogniSight SDK does not handle the image acquisition and display part. It just provides for making a copy of a pixel array to a dedicated buffer we call the CogniSight buffer. Depending on your application, this pixel 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.

PACKAGE CONTENT

Several versions of the DLL exist and are targeted at the following NeuroMem hardware platforms:

- CogniSight_NSnK.dll, interfaces to the digital neurons of the NeuroStack board, or a stack of them, or a chain of simulated neurons with different capacity
- CognSight_Simu.dll, simulates networks of different capacity

Bin Folder

All DLLs have the same entry points.
CogniPat_Simu.dll has no dependencies.
CogniPat_NSnK.dll requires that the FTDI USB driver for the NeuroStack interface.

- Win32/ CogniSight_Simu.dll, CogniSight_NSnK.dll
- x64/ CogniSight_Simu.dll, CognSight_NSnK.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
- CogniSghtClass.m Class defining the entry points to the DLL for MatLab interface

Examples

Examples are provided to helps understand how to use the neurons to learn and recognize visual objects or events. Depending on your hardware, each example can be executed with the CogniSight_Simu.dll or CogniSight_NSnK.dll. This selection is made as follows depending on the programming language:

	Where do I select the CogniSight DLL?	**Location**
C++	Linked library	Project/Properties/Linker/Input (declared in C++ project)
C#	DLL name in DLL Import	CogniSight.cs (included in your C# project)
MatLab	DLL path in the CogniSight Class method	CogniSightClass.m (instantiated in your MatLab script)

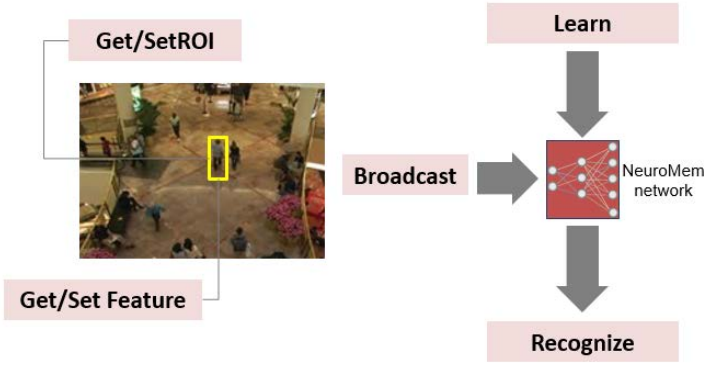
CONCEPTS AND DEFINITIONS

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 has been acquired by a CogniSight Sensor in which case it resides automatically in the CogniSight image memory frame.

Region of Interest

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.

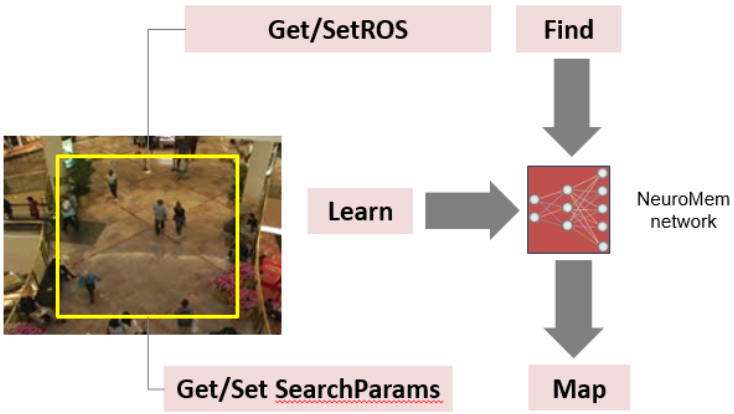


The classification of the ROI can be based on a single or multiple features (or signatures). In the case of multiple features, the context of the neurons must be changed for each type of feature. The context value can be assigned to a feature identification number for example starting at the value 1 (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 has to 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.

Region of Scan

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.



Visual Objects

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.

Maps

A Map **(MAP)** are 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.

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.

Project

A project file is more complete than just a knowledge file because it describes exactly how the knowledge was built (the network settings, the experts and their feature extractions, the user preferences such as the names of the categories and their color tags). The project file includes all the necessary information to expand an existing knowledge and to use it for immediate recognition of still or live images, produce meta data, selective recording and more.

Feature Extraction

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.

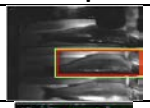
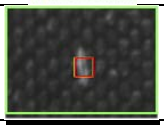
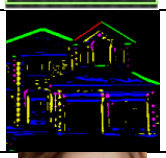
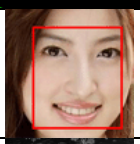
Multiple Contexts

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.

Deep Learning

The NeuroMem neurons can be an enabler for fast "deep learning": A 1st subnetwork is trained to recognize sparse codes in an image and produce a map of the recognized primitive categories with a smaller resolution. A 2nd subnetwork is then trained to recognize the patterns of primitive categories and so on as necessary to come up with a final decision.

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	

HARDWARE INTERFACE FUNCTIONS

int Version();

Returns the version of the DLL.

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 definition
0= Simulation of NeuroMem neurons	Code for the simulated device: 0 = CM1K 1 = NS4K_chain of 4096 2 = CurieNeurons 3 = NM500
1=NeuroStack board (4096 neurons of 256 bytes)	N/A. Default is 0.
2= V1KU camera (1024 neurons of 256 bytes)	DeviceID is the USB device number connected to your host. Default is 0.

int Disconnect();

Closes the communication with the current device.

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.

int CountNeuronsandReset();

Detect the number of neurons available in the selected platform and clear their content including registers and memory. This function should only be executed at the launch of an application.

Remark: The execution of this function is only necessary when using a NeuroMem platform with an expansion connector and therefore a variable capacity of neurons. Its execution takes approximately 1 second per thousand neurons.

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.

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.

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

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

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.

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.

void SetROI(int Width, int Height)

Defines the nominal width and height of the current ROI.

void GetROI(int *Width, int *Height)

Reads the nominal width and height of the current ROI.

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

void GetFeat(int *FeatID)

Set the feature extraction method in use.

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	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 Appendix A for more details on the feature extractions.

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)

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.

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.

REGIONS OF SEARCH (ROS)

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.

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

Reads the current Region Of Search

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. 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.

LEARNING FUNCTIONS

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 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.
    - The Save and Restore mode is not compatible with the learning mode.

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.

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:

- 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.

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.

RECOGNITION FUNCTIONS

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.

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.

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

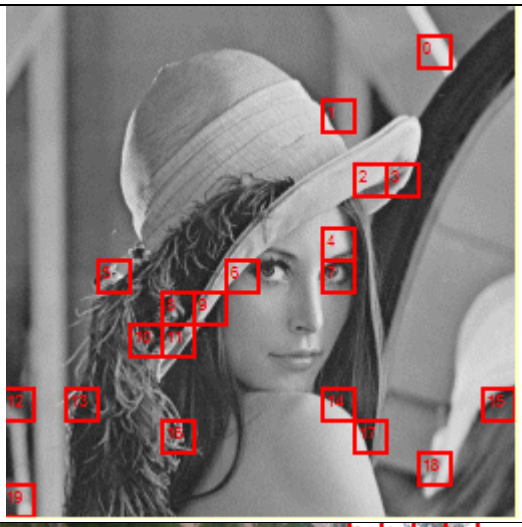
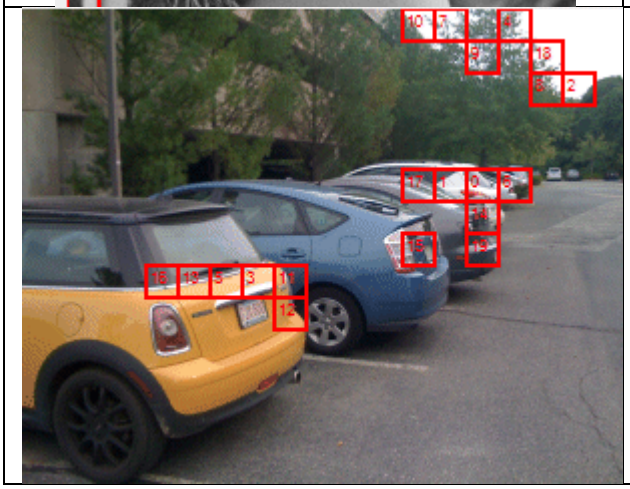
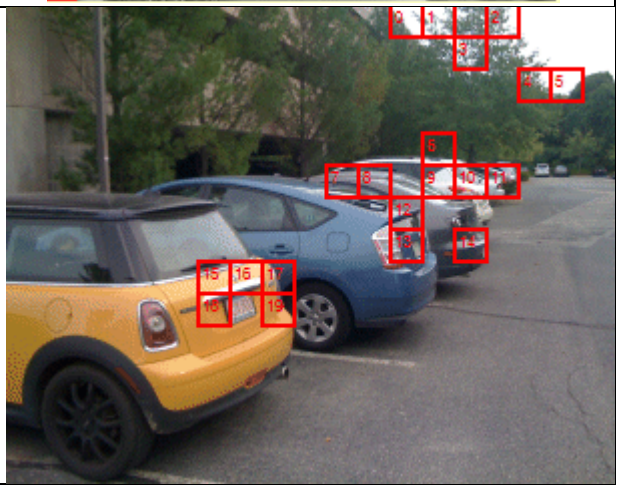
Recognizes the feature vectors extracted from the Region of Search scanned in a raster displacement with horizontal stepX and vertical stepY. The output arrays describe all the recognized objects including their X and Y position in the image, and the distance, category and neuron identifier of the closest firing neuron. The skip_if_reco flag allows to optimize the scanning process by stepping directly to the next adjacent ROI in the case of a positive recognition.

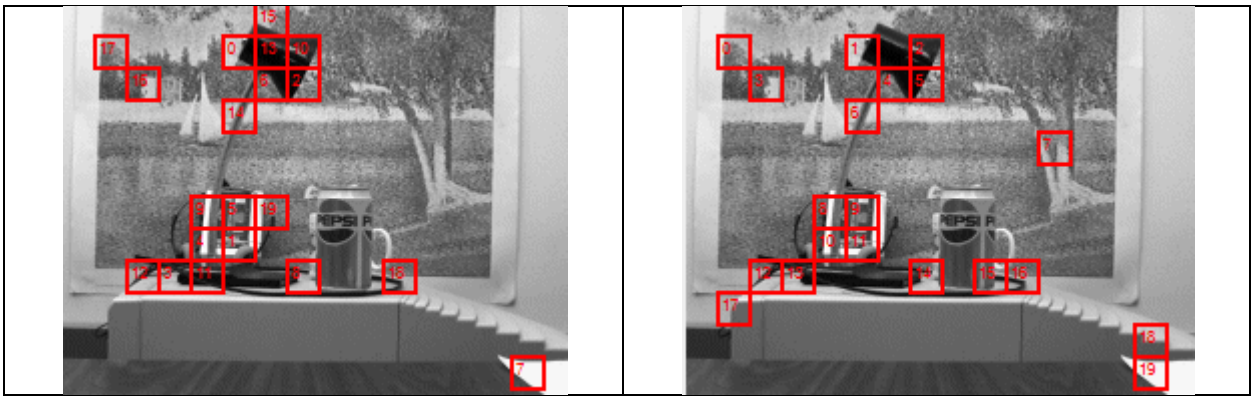
int FindROSAnomalies(int stepX, int stepY, int MaxNbr, int *Xpos, int *Ypos)

Reports the feature vectors extracted from the Region of Search scanned in a raster displacement with horizontal stepX and vertical stepY and which are not recognized. The output arrays Xpos and Ypos report the positions of these unknown locations.

FindROS Saliencies

Two functions are available in the SDK as described below:

Method 1	Method2
Return the K first salient blocks in an image	Return the K most unique blocks in an image
Learn all blocks, forcing the commitment of one neuron per block.	Learn blocks only if not recognized, iterative with adjustment of Maxif until the count of requested unique blocks is reached.
Requires a fixed and significant amount of neurons equal to the number of steps to scan the ROS	Uses less neurons, but takes longer due to iterations.
The output lists are the positions of the first K blocks sorted in decreasing order of saliency. This can still include blocks with a poor saliency of K is a generous value. Some survey of the AIF list can help detect when saliency becomes unsignificant.	The output list are the positions of the K most unique blocks in the ROS
	
	



void FindROSSalientBlocks(int stepX, int stepY,int K, int *CtrX, int *CtrY, int *AIF)

Learn all the blocks of pixels of the ROS and analyze the Active Influence Fields (AIFs) of the committed neurons to identify the blocks which are the most salient. Return the center position of the first K blocks with their associated highest AIFs.

Remark1: The higher the influence field of a neuron, the more salient the pattern held in its memory since we can assume that this neuron is among the least which had to shrink their influence field during the learning due to similar blocks.

Remark2: Adjacent blocks belonging to a uniform background area have similar feature vectors. Learning them with a different category forces the commitment of one neuron per block but their influence fields are very small, if not set to the minimum, since the patterns are redundant.

Remark3: The contents of the neuron represent a tiling of the ROS.

- The size of the block is the size of the current ROI and its default should be a block of 16x16 pixels, but it is possible to execute the function on larger or smaller block size.
- The feature used during the learning is the current feature extraction. Its default should be a monochrome or color subsample (featID=0 or featID=3), but the function will actually execute with any feature extraction.
- The category of the committed neurons is the linear index of the block within the ROS.

Warning, this function clears the content of the neurons prior to its execution.

void FindROSUniqueBlocks(int stepX, int stepY,int K, int *CtrX, int *CtrY)

Build a codebook of the blocks in the ROS changing the value of the Maximum Influence Field at each iteration to converge towards a number of unique blocks equal to K. Return the center position of the K unique blocks.

Remark1: Adjacent blocks belonging to a uniform background area have similar feature vectors. Learning them should commit a single neuron which will fire several times during the learning. A block is unique if no other block falls within the influence field of its neuron.

- The size of the block is the size of the current ROI and its default should be a block of 16x16 pixels, but it is possible to execute the function on larger or smaller block size.
- The feature used during the learning is the current feature extraction. Its default should be a monochrome or color subsample (featID=0 or featID=3), but the function will actually execute with any feature extrcation.

- The category of the committed neurons is the linear index of the block within the ROS.

Warning, this function clears the content of the neurons prior to its execution. It restores the value of the MAXIF at the end.

TRANSFORM FUNCTIONS

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.

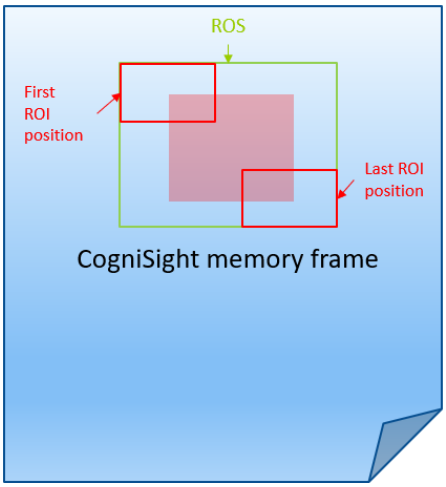
The output arrays CatMap, DistMap and NidMap can then be mapped into images with a width of MapWidth for a 2D mapping of the categories, distances and firing neurons in the region of search. The amplitude of these values can be much greater than intensity values ranging between [0-255] and scaling might be required.

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

The function returns the width of the resulting map, or the number of samples which constitute a row. This number is useful to display the output array in a 2D plane.

Warning:
The output consists of 3 arrays of same dimension L: Distances[L], Categories[L], Nids[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



The interpretation of the output Maps may require their conversion into pseudo-color images (8-bit or higher) and displayed with appropriate color lookup table.

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

KNOWLEDGE LEVEL FUNCTIONS

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

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.

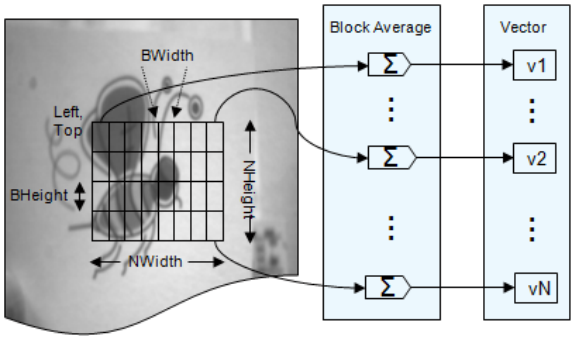
Among the most commonly used functions:

- AvailableNeurons
- CommittedNeurons
- ClearNeurons

APPENDIX A: FEATURE EXTRACTIONS

SubSample

Subsample is a vector which appends the average intensity of blocks of pixels extracted from the region of interest. The region must contain less than 256 blocks so the output vector fits on 256 bytes. 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 region with a size [NWidth, NHeight] is divided into up to 256 blocks of size [BWIDTH, BHEIGHT].
- 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*BWIDTH
    - NHEIGHT= m*BHEIGHT
    - n*m <=256.

Histogram

Histogram is a vector which gives the distribution of the grey-level values in the region of interest. If the image in the CogniSight memory plane is color and encoded with 3 bytes per pixel, the histogram reports the distribution of the average of Red, Green and Blue per pixel.

The number of bins in the case of a grey-level histogram is 256. The amplitude of the histogram is scaled by multiplying it by the ratio (255/N) with N, the total number of pixels in the region of interest.

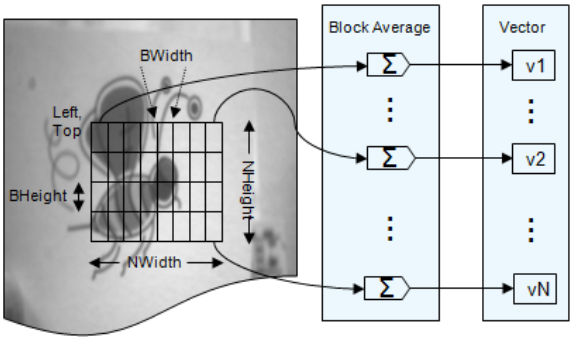
This feature vector gives an indication of the number of different shades in the region, the presence of noisy pixels and more.

Histogram Cumulative

The 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.

SubsampleRGB

SubsampleRGB is a vector composed of the average Red intensities of blocks of pixels extracted from the region of interest, followed by the average Green and Blue intensities of these same blocks The blocks are surveyed in a raster displacement. The region must contain less than 81 blocks so the output vector fits on 81x3= 243 bytes. 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 region with a size [NWidth, NHeight] is divided into up to 256 blocks of size [BWIDTH, BHEIGHT].
- 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*BWIDTH
    - NHEIGHT= m*BHEIGHT
    - n*m <=81

Histogram RGB

Histogram 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.

If the image in the CogniSight memory plane is monochrome and encoded with 1 byte per pixel, the histogramRGB will be a succession of 3 identical histograms since the Red, Green and Blue intensity of each pixel are the equal.

The number of bins in the case of a color histogram is 256. The amplitude of the histogram is scaled by multiplying it by the ratio (255/N) with N, the total number of pixels in the region of interest.

This feature vector gives an indication of the number of different shades in the region, the presence of noisy pixels and more.

Histogram RGB Cumulative

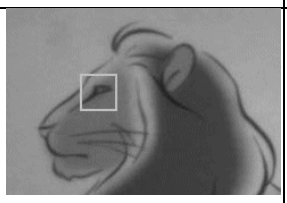
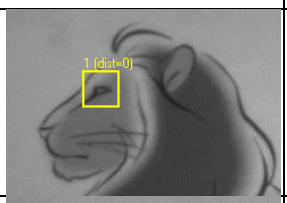
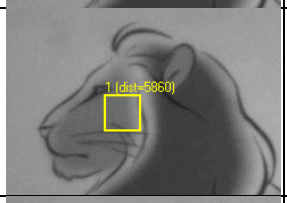
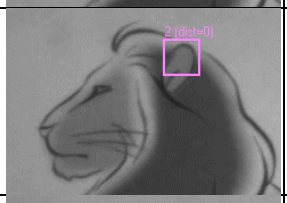
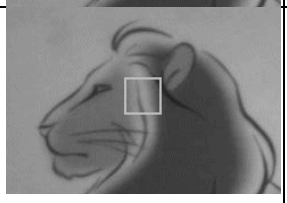
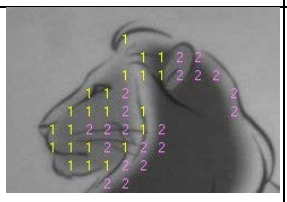
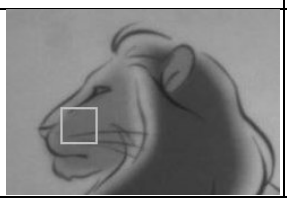
The cumulative histogram is a mapping of the standard histogram which counts the cumulative number of pixels in all of the bins up to the specific bin.

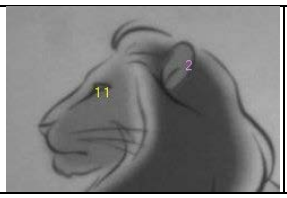
Horizontal, Vertical or Composite Profiles

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.

APPENDIX B: TUTORIAL, HINTS AND TIPS

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.

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)
```

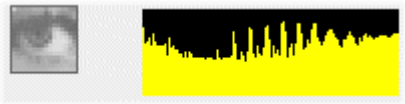
Example2: Recognize a same example using 2 feature vectors

```
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!")
```

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 

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.

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)
```

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)
```

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 retun(“Pass”)
```

Surface Inspection

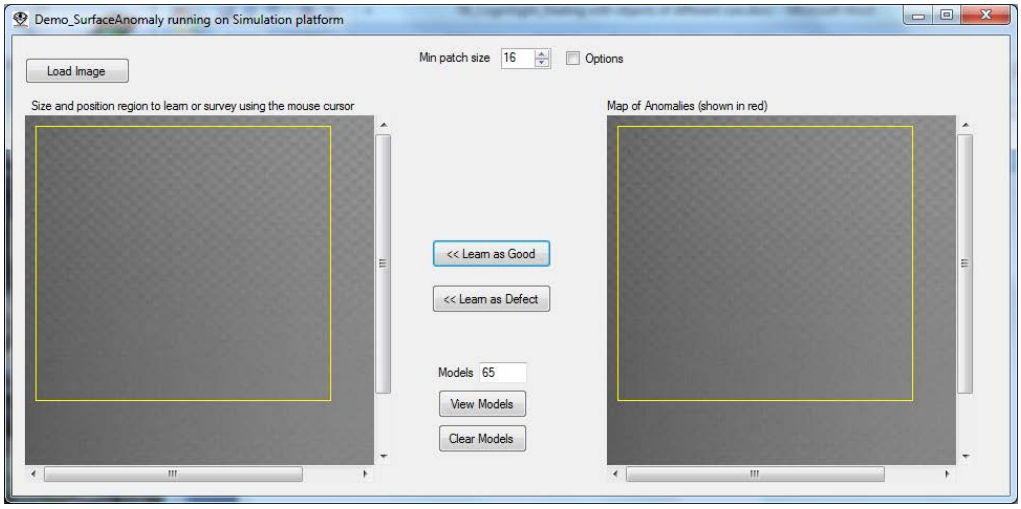
Texture learning is easy with the CogniSight engine. A region of interest can be divided into patches and the neurons will automatically learn the patches which are significant to describe the texture of the region.

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.

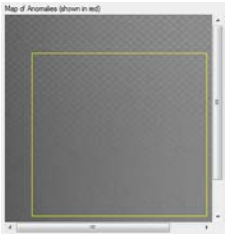
Example patches of 16x16 pixels 

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.

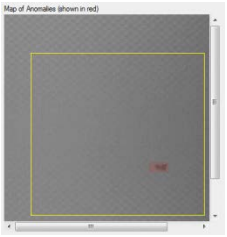


The image to the right is supposed to highlight the patches which are not recognized by the CogniSight engine because they do not match any of the 65 models. In this case, all learned patches are positively identified.



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.