Change Detection with Weightless Neural Networks

Massimo De Gregorio
Istituto di Cibernetica
"E. Caianiello" (ICIB–CNR)
Via Campi Flegrei, 34
80078 Pozzuoli, ITALY
massimo.degregorio@cnr.it

Maurizio Giordano
Istituto di Calcolo e Reti
ad Alte Prestazioni (ICAR–CNR)
Via P. Castellino, 111
80131 Naples, ITALY
maurizio.giordano@cnr.it

Abstract

In this paper a pixel–based Weightless Neural Network (WNN) method to face the problem of change detection in the field of view of a camera is proposed. The main features of the proposed method are 1) the dynamic adaptability to background change due to the WNN model adopted and 2) the introduction of pixel color histories to improve system behavior in videos characterized by (des)appearing of objects in video scene and/or sudden changes in lightning and background brightness and shape. The WNN approach is very simple and straightforward, and it gives high rank results in competition with other approaches applied to the ChangeDetection.net 2014 benchmark dataset.

1. Introduction

The ChangeDetection.net (CDNET) 2014 competition invites academies and industries to publish results of their more advanced change&motion detection (CD) methods and techniques. As in the previous CDNET 2012 challenge, competing solutions would be classified according to different background modeling approaches, from statistical (KDE, single and mixture of Gaussian), to clustering models, from computer vision techniques (median or histogram analysis) to neural network modeling.

Up to now, previously proposed neural network approaches to CD problem falls in the class of weighted neural network systems [8][7]. On the contrary, our approach, which is called CwisarDH and extends a previous method [4] still competing in the CDNET 2012 challenge, relies on a weightless neural network architecture named WiSARD [3].

In this paper[1] the proposed CwisarDH method to CD problem is described by emphasizing its main characteristics: 1) pixel–based processing without the need of neighborhood information; 2) the simplicity of pre– and post–processing of video data; 3) straightforward use of a WNN for the purpose without *ad hoc* modifications.

The paper is so organized: in Section 2 the adopted WNN model is introduced; in Section 3 the proposed WNN–based approach to change detection is presented together with the experimental settings; Section 4 reports and discusses the experimental results of CwisarDH detection capabilities when running on the CDNET 2014 video dataset; finally, Section 5 summarizes concluding remarks and future perspectives.

[1]This research work was partially funded by the EU FP7-ICT-2012-8 under the MIDAS Project – Grant Agreement no. 318786.

2. The WiSARD weightless neural model

Weightless neural networks are based on networks of Random Access Memory (RAM) nodes [1]. As illustrated by Figure 1, a RAM–based neuron/node is capable of recognizing n bit inputs (n–tuple) coming from the retina (usually a black and white image). The WNNs have a basis for their biological plausibility because of the straightforward analogy between the address decoding in RAMs and the integration of excitation and inhibitory signaling performed by the neuron dendritic tree. WiSARD systems are a particular type of WNN. While the use of n–tuple RAM nodes in pattern recognition problems is old, dating about

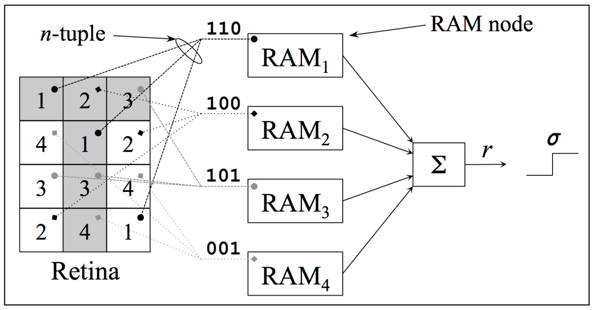

Figure 1. A WiSARD discriminator.

60 years, with the availability of integrated circuit memories in the late 70s, the WiSARD (**W**ilkes, **S**tonham and **A**leksander **R**ecognition **D**evice) was the first artificial neural network machine to be patented and produced commercially [3]. The WiSARDs can be, in fact, developed directly on reprogrammable hardware. This characteristic finds a concrete applicability in embedded robotic systems.

In the WiSARD model, RAM input lines are connected to the retina by means of a biunivocal pseudo–random mapping as a set of uncorrelated n–tuples. For instance, in Figure 1, the n–tuple, and so the memory address of RAM_1, is always formed by the colors of the 3 pixel labeled with "1". In our example and for the "T" represented on the retina, RAM_1 will receive as input the tuple `110`. Each n–tuple is used as a specific address of a RAM node memory location, in such a way that the input pattern is completely mapped to a set of RAM locations.

A WiSARD *discriminator*, composed by m RAM–based neurons, is trained with representative data of a specific class/category. In order to use the network as a discriminator, one has to set all RAM memory locations to '0' and choose a training set formed by binary patterns of $(m \times n)$ bits. For each training pattern, a '1' is stored in the memory location of each RAM addressed by this input pattern. Once the training of patterns is completed, RAM memory contents will be set to a certain number of '0's and '1's. The information stored by RAM nodes during the training phase is used to deal with previous unseen patterns. When one of these is given as input, RAM memory contents addressed by the input pattern are read and summed by the summing device Σ. The number r thus obtained, which is called the *discriminator response*, is equal to the number of RAMs that output '1'. It is easy to see that r necessarily reaches the maximum m if the input pattern belongs to the training set. r is equal to zero if no n–bit component of the input pattern appears in the training set (not a single RAM outputs '1'). Intermediate values of r express a kind of "similarity measure" of the input pattern with respect to the patterns in the training set. The summing device enables this network of RAM nodes to exhibit – just like other ANN models based on synaptic weights – generalization and noise tolerance [2].

3. The CwisarDH approach to CD

Algorithm 1 describes the CwisarDH method pseudocode. In order to feed the discriminators with the right input, CwisarDH creates one discriminator for each pixel of the video frame. The RGB color of the pixel is represented by a binary (black & white) image, where the columns represent the color channel (R, G and B) and the rows the color channel values (see Figure 2). CwisarDH adopts 192 values (that is, the retina size is 192×3) to represent the channel values. This is the value the system works at the best (the

```
Input: video
Output: outvideo (B&W video with detected moving objects)
while getting a new frame from video do
    if frame belogs to trainset then
        foreach pixel in frame do
            train the pixel discriminator with RGB encoding;
            set pixel as bg in outframe;
    else
        foreach pixel in frame do
            use RGB encoding to get response from pixel
            discriminator;
            if response > $\sigma$ then
                empty pixel history buffer;
                train the pixel discriminator with RGB encoding;
                set pixel as bg in outframe;
            else
                if pixel history buffer is full then
                    re-train the pixel discriminator with RGB
                    encodings stored in the pixel history buffer;
                    empty pixel history buffer;
                else
                    store RGB encoding in pixel history buffer;
                set pixel as fg in outframe;
```

Algorithm 1. CwisarDH method pseudocode

less is the value the faster is the system). Other two parameters have been fixed to face the CD challenge: RAM address memory and threshold σ. With 16 bit address location and 86% as threshold the average performance of the system is the best.

The system parameters are constrained to the application domain. For instance, in case of dynamic backgrounds the system can better face the problem with threshold values around 80%. This is because it can absorb and better classify shimmering water or trees shaken by the wind.

CwisarDH is trained on a certain number of pixel instances taken in different frames of the video under examination. After the training phase, the system classifies the pixel as belonging to the background only if the corresponding discriminator response is greater than the fixed threshold σ, otherwise the pixel is considered belonging to the foreground. The system takes the correctly classified pixels to further train the associated discriminator: the on–line training is a peculiar characteristic of weightless systems. In this way, CwisarDH adapts itself both to dynamic backgrounds and to gradual changes in light.

CwisarD**H** extends the previous method CwisarD [4] by introducing a pixel classification **H**istory support: a k–sized

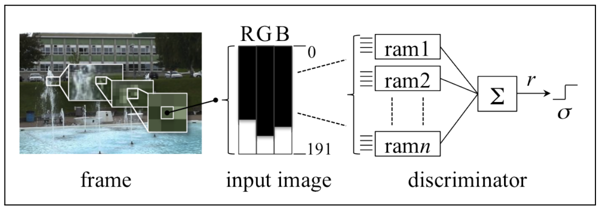

Figure 2. CwisarDH input encoding

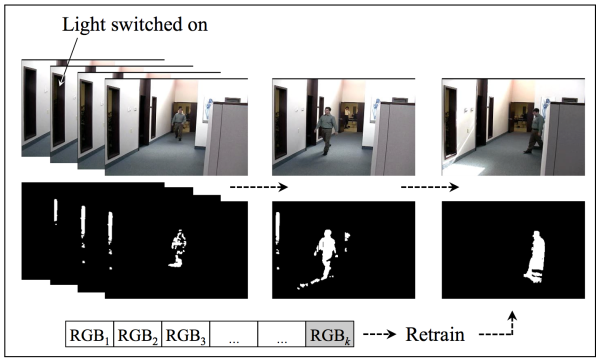

Figure 3. CwisarDH retrain on new pixel background

buffer is associated to each pixel to stores pixel colors continuously classified as foreground in k successive frames. When the buffer is full, the color history is used to reset and then to train the associated discriminator on buffered data (see Figure 3). On the contrary, each time the pixel is classified as background the history is emptied.

The history buffer support was introduced to improve performance of the previous CwisarD system, especially to face with both the case of intermittent objects, like (des)appearing of objects that change status from background to foreground in the scene, and the case of sudden changes in light, shape and colors of background regions (like in *Bad Weather* and *Turbulence* dataset categories).

As an example, Figure 3 shows the case of a light switched on and left on for all the video duration. Because the corresponding pixels are continuously classified as foreground in the successive k frames, the buffer gets full and the discriminator is retrained on the buffered RGB instances. From this point on, the pixels representing the switched light are absorbed and considered as part of the background.

The result of CwisarDH is displayed after the application of two post–processing filters: erosion and dilation. This is to reduce the salt and pepper effect in the output video frames.

3.1. CwisarDH parallelism on multicores

CwisarDH is implemented in C++ and uses the OpenCV library [9] for image pre/post–processing and visualization. CwisarDH software is characterized by a high degree of potential parallelism, since pixel–based computation in a frame has neither data nor control dependency on other pixel computations in the same frame. In fact, one WiSARD discriminator is associated to each pixel and trained by pixel values gathered in successive frame of the timeline. While computation on each pixel of the same video frame can be parallelized, synchronization is require at each new frame. For this reason we implemented an OpenMP C++ version of CwisarDH to better exploit parallelism on a multicore CPUs. We used the `parallel for` OpenMP [10] directive to parallelize the loop iterating on frame pixels. This directive forces the compiler to generate threads[2] acting in parallel on separated regions of the image frame.

	No. of Threads	320×240 pixels	720×480 pixels
		Frame Rate (in fps)	
Sequential			
no optimization	9	5.25	1.38
optimization lvl. 02	9	7.45	2.23
OpenMP with opt. lvl. 02			
schedule(dynamic,1)	16	18.5	4.22

Table 1. CwisarDH OpenMP vs Sequential timing

We carried out timing measurements on a 3.4 Ghz Intel Core i7 (quadcore) with 8GB RAM and Mac OS X v.10.9.2 operating system to compare the OpenMP version of CwisarDH with the sequential one. The results are reported in Table 1. We measured the mean value of video frame processing rate over one hundred frames soon after CwisarDH starts classifying. It is worth noticing the significant speed up gained in both resolutions: the number of threads increases, and, much more interesting, they exploit the multicores more efficiently.

4. CwisarDH results evaluation

Some snapshots of the system outputs are reported in Figure 4. Table 2 reports system results on all videos in the dataset while Table 3 reports the average measures of all CDNET 2014 competing methods.

As one can notice, the system behaves quite well in most of the situations. This is due both 1) to the characteristic of artificial neural networks that well adapt to background changing and 2) to the pixel colors history buffer support proposed in the new method. In fact, being based on an artificial neural network paradigm, CwisarDH gives the best results (first in the *Average ranking*) on the videos belonging to the category *Camera Jitter* and very good results on videos belonging to *PTZ* and *Dynamic Background* where one has to face the problem of, for instance, continuous change of the background or shimmering water or waving trees. On the other side, the introduction of the pixel colors buffer allows the system to dial even with situations in which the original background changes because of the (des)appearance of an object in the scene (like in the *Intermittent Object Motion* dataset category). This is based on the absorption in the new background of persistent pixels continuously classified as foreground in k consecutive video frames. The overall best metric values obtained by CwisarDH are reported in Table 3 with black cells.

[2]The number of threads is chosen by the C++ runtime and it depends on several dynamic parameters, such as the OS version and current load, the user environment settings, and so on.

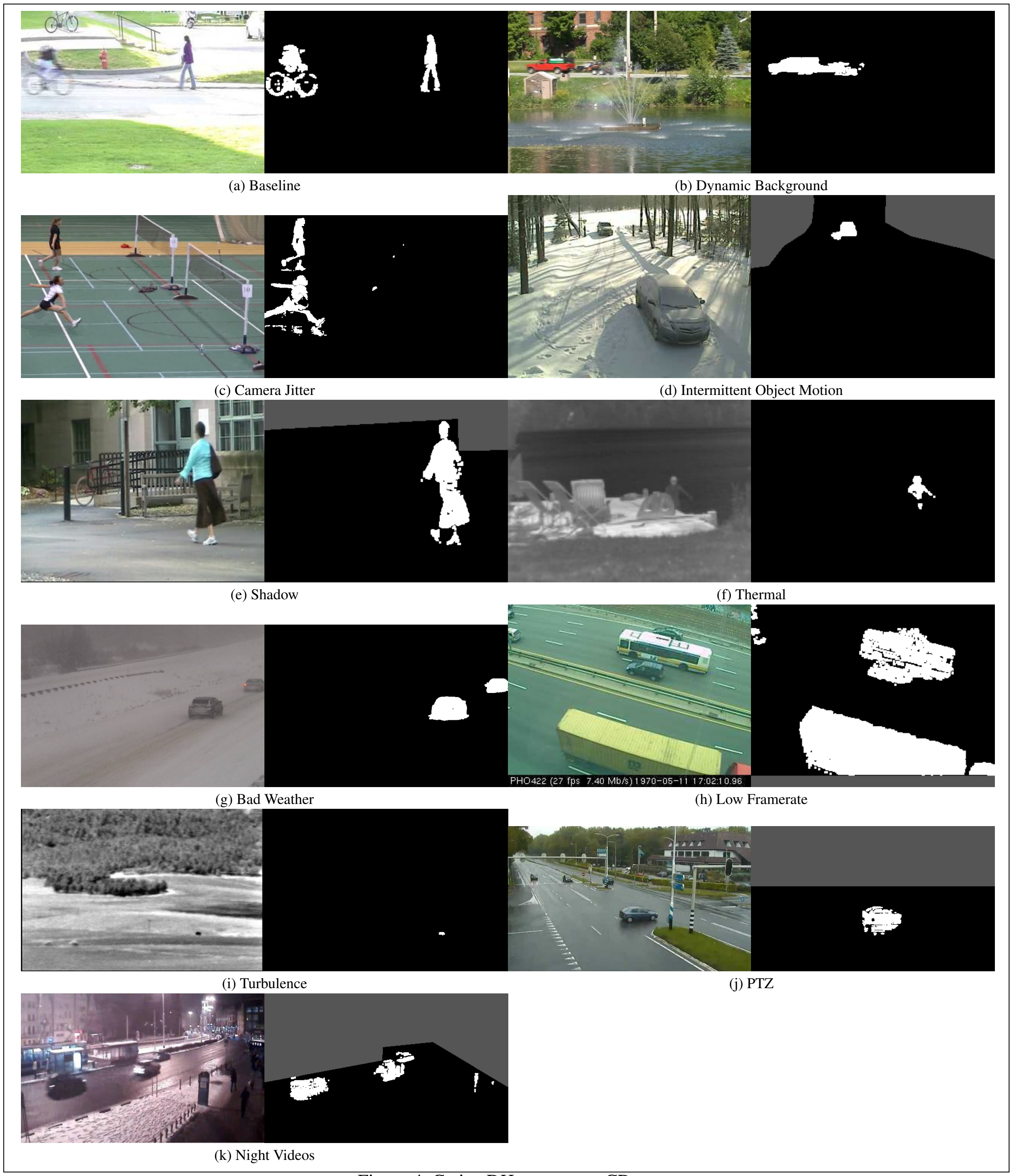

Figure 4. CwisarDH outputs on CDnet

5. Conclusions

CwisarDH is a method based on Weightless Neural Networks to face the change&motion detection problem in videos. CwisarDH outperforms other competitors in the CDNET 2014 challenge. The main features of CwisarDH are: 1) the dynamic adaptability to background change due to the WiSARD model adopted; 2) the use of pixel color history buffers to improve the system behavior in videos characterized by (des)appearing of objects in the scene and slow/fast changes in lightning and background brightness.

	Ranking across categories	Ranking	Recall	Specificity	FPR	FNR	PWC	F–measure	Precision
FTSG (Flux Tensor with Split Gaussian models)	1.64	2.00	0.7657	0.9922	0.0078	0.2343	1.3763	0.7283	0.7696
SuBSENSE	3.00	4.43	0.7842	0.9742	0.0258	0.2158	3.3712	0.6889	0.7135
CwisarDH	**3.45**	**4.57**	**0.6608**	**0.9948**	**0.0052**	**0.3392**	**1.5273**	**0.6812**	**0.7725**
Spectral-360	4.36	4.43	0.7345	0.9861	0.0139	0.2655	2.2722	0.6732	0.7054
Bin Wang Apr 2014	6.27	5.57	0.7035	0.9794	0.0206	0.2965	2.9009	0.6577	0.7163
KNN	6.55	7.00	0.6650	0.9802	0.0198	0.3350	3.3200	0.5937	0.6788
SC_SOBS	7.64	7.57	0.7621	0.9547	0.0453	0.2379	5.1498	0.5961	0.6091
KDE - ElGammal	8.64	9.71	0.7375	0.9519	0.0481	0.2625	5.6262	0.5688	0.5811
Mahalanobis distance	9.00	8.14	0.1644	0.9931	0.0069	0.8356	3.4750	0.2267	0.7403
GMM | Stauffer & Grimson	9.27	8.14	0.6846	0.9750	0.0250	0.3154	3.7667	0.5707	0.6025
CP3-online	9.82	8.43	0.7225	0.9705	0.0295	0.2775	3.4318	0.5805	0.5559
GMM | Zivkovic	10.18	10.71	0.6604	0.9725	0.0275	0.3396	3.9953	0.5566	0.5973
Multiscale Spatio-Temporal BG Model	11.45	12.00	0.6621	0.9542	0.0458	0.3379	5.5456	0.5141	0.5536
Euclidean distance	13.00	12.29	0.6803	0.9449	0.0551	0.3197	6.5423	0.5161	0.5480

Table 3. Average measures comparison among all methods

		Recall	Specificity	FPR	FNR	PWC	Precision	F-measure
Baseline	pedestrians	0.9681	0.9995	0.0005	0.0003	0.0766	0.9546	0.9613
	PETS2006	0.8084	0.9985	0.0015	0.0025	0.3968	0.8766	0.8411
	office	0.8898	0.9989	0.0011	0.0082	0.8603	0.9840	0.9346
	highway	0.9225	0.9949	0.0051	0.0049	0.9379	0.9195	0.9210
Dynamic Background	overpass	0.8285	0.9997	0.0003	0.0023	0.2563	0.9766	0.8965
	canoe	0.8979	0.9994	0.0006	0.0037	0.4215	0.9815	0.9378
	fall	0.8430	0.9926	0.0074	0.0028	1.0065	0.6722	0.7480
	fountain02	0.9184	0.9999	0.0001	0.0002	0.0296	0.9423	0.9302
	fountain01	0.6382	0.9996	0.0004	0.0003	0.0673	0.5872	0.6116
	boats	0.7604	0.9997	0.0003	0.0015	0.1810	0.9394	0.8405
Camera Jitter	boulevard	0.6031	0.9943	0.0057	0.0195	2.4093	0.8382	0.7015
	sidewalk	0.7138	0.9987	0.0013	0.0077	0.8734	0.9360	0.8100
	badminton	0.8079	0.9948	0.0052	0.0068	1.1608	0.8466	0.8268
	traffic	0.8498	0.9846	0.0154	0.0100	2.3798	0.7855	0.8164
Intermittent Obj. Motion	abandonedBox	0.2984	0.9959	0.0041	0.0354	3.7621	0.7872	0.4327
	winterDriveway	0.4999	0.9860	0.0140	0.0038	1.7665	0.2119	0.2976
	sofa	0.8294	0.9955	0.0045	0.0078	1.1745	0.8940	0.8605
	tramstop	0.1625	0.9971	0.0029	0.1832	15.2681	0.9241	0.2764
	parking	0.6017	0.9729	0.0271	0.0334	5.5843	0.6501	0.6250
	streetLight	0.9376	0.9992	0.0008	0.0032	0.3806	0.9832	0.9598
Night Videos	tramStation	0.5107	0.9925	0.0075	0.0138	2.0761	0.6577	0.5749
	busyBoulvard	0.1738	0.9971	0.0029	0.0302	3.1947	0.6877	0.2775
	streetCornerAtNight	0.7249	0.9880	0.0120	0.0014	1.3339	0.2306	0.3499
	fluidHighway	0.5955	0.9724	0.0276	0.0058	3.2948	0.2363	0.3384
	winterStreet	0.5394	0.9778	0.0222	0.0141	3.5153	0.4262	0.4762
	bridgeEntry	0.1622	0.9969	0.0031	0.0121	1.4980	0.4266	0.2350
Thermal	lakeSide	0.4540	0.9981	0.0019	0.0107	1.2373	0.8198	0.5844
	park	0.6417	0.9976	0.0024	0.0074	0.9610	0.8482	0.7306
	diningRoom	0.7199	0.9945	0.0055	0.0263	2.9048	0.9253	0.8098
	library	0.9357	0.9898	0.0102	0.0154	2.0642	0.9563	0.9459
	corridor	0.8825	0.9944	0.0056	0.0040	0.9324	0.8434	0.8625
PTZ	twoPositionPTZCam	0.7215	0.9956	0.0044	0.0043	0.8586	0.7152	0.7184
	zoomInZoomOut	0.5275	0.9941	0.0059	0.0010	0.6887	0.1587	0.2440
	continuousPan	0.2157	0.9978	0.0022	0.0050	0.7066	0.3866	0.2769
	intermittentPan	0.0683	0.9996	0.0004	0.0133	1.3513	0.7290	0.1249
Turbulence	turbulence2	0.8889	1.0000	0.0000	0.0000	0.0046	0.9850	0.9345
	turbulence3	0.7140	0.9996	0.0004	0.0047	0.5035	0.9638	0.8203
	turbulence0	0.6980	0.9999	0.0001	0.0006	0.0658	0.9362	0.7998
	turbulence1	0.5479	0.9993	0.0007	0.0017	0.2379	0.7592	0.6365
Shadow	copyMachine	0.8705	0.9917	0.0083	0.0096	1.6663	0.8869	0.8786
	bungalows	0.9627	0.9763	0.0237	0.0024	2.4489	0.7218	0.8250
	busStation	0.8676	0.9924	0.0076	0.0051	1.2176	0.8145	0.8402
	peopleInShade	0.9666	0.9889	0.0111	0.0020	1.2325	0.8394	0.8985
	backdoor	0.8357	0.9997	0.0003	0.0033	0.3576	0.9817	0.9028
	cubicle	0.7686	0.9971	0.0029	0.0046	0.7392	0.8413	0.8033
Bad Weather	skating	0.8345	0.9996	0.0004	0.0086	0.8586	0.9905	0.9058
	wetSnow	0.2834	0.9997	0.0003	0.0094	0.9553	0.9195	0.4332
	snowFall	0.7236	0.9992	0.0008	0.0022	0.2945	0.8816	0.7948
	blizzard	0.8372	0.9986	0.0014	0.0019	0.3258	0.8777	0.8570
Low Framerate	tunnelExit_0_35fps	0.6065	0.9960	0.0040	0.0111	1.4717	0.8090	0.6932
	port_0_17fps	0.4118	0.9999	0.0001	0.0002	0.0292	0.5105	0.4559
	tramCrossroad_1fps	0.8279	0.9932	0.0068	0.0049	1.1380	0.7768	0.8015
	turnpike_0_5fps	0.8175	0.9904	0.0096	0.0147	2.2504	0.8721	0.8439

Table 2. CwisarDH results on CDnet

In many real situations, there is no opportunity to have a certain number of frames representing the background (busy highway, underground stations, ...). In these cases, the system cannot be appropriately trained and its performance degrades. To overcome this problem, we are going to adopt a self–adaptive version of CwisarDH that already gave very good results in the problem of tracking deformable objects [6][5]. This new system version does not need to be trained in advance and it is able to dynamically generate the background model very quickly.

References

[1] I. Aleksander, M. De Gregorio, F. M. G. França, P. M. V. Lima, and H. Morton. A brief introduction to weightless neural systems. In *ESANN 2009*, pages 299–305, 2009.

[2] I. Aleksander and H. Morton. *An introduction to neural computing*. Chapman & Hall, London, 1990.

[3] I. Aleksander, W. V. Thomas, and P. A. Bowden. WiSARD a radical step forward in image recognition. *Sensor Review*, 4:120–124, 1984.

[4] M. De Gregorio and M. Giordano. A WiSARD-based approach to CDnet. In *Proc. of 1st BRICS Countries Congress (BRICS-CCI)*, 2013.

[5] M. De Gregorio, M. Giordano, S. Rossi, and M. Staffa. Can you follow that guy?. In *ESANN 2014*, pages 511–516, 2014.

[6] M. De Gregorio, M. Giordano, S. Rossi, and M. Staffa. Tracking deformable objects with WiSARD networks. In *Workshop on Deformable Object Manipulation – INNOROBO2014*, 2014.

[7] S. Ghosh, M. Roy, and A. Ghosh. Semi–supervised change detection using modified self–organizing feature map neural network. *Applied Soft Computing*, 15(0):1 – 20, 2014.

[8] L. Maddalena and A. Petrosino. The SOBS algorithm: What are the limits?. In *CVPR Workshops*, pages 21–26, 2012.

[9] OpenCV. Open source computer vision. http://www.opencv.org.

[10] OpenMP. The openmp api specification for parallel programming. http://www.openmp.org.