Competitive Neural Networks Applied to Face Localization

Technical Report B-13-03

Erik V. Cuevas[1,2], Daniel Zaldívar[1,2], Raúl Rojas[1]

[1] Freie Universität Berlin, Institut für Informatik
Takustr. 9, D-14195 Berlin, Germany

[2] Universidad de Guadalajara
Av. Revolución No. 1500, C.P. 44430, Guadalajara, Jal., México

`{cuevas, zaldivar, rojas}@inf.fu-berlin.de`

June 8, 2004

Abstract

Color-segmentation is very sensitive to changes in the intensity of light. Many algorithms do not tolerate variations in color hue which correspond, in fact, to the same object. Learning Vector Quantization (LVQ) networks learn to recognize groups of similar input vectors in such a way that neurons physically near to each other in the neuron layer respond to similar input vectors. Learning is supervised, the inputs vectors into target classes are chosen by the user. In this work a new algorithm based on LVQ is presented. It involves neural networks that operate directly on the image pixels with a decision function. This algorithm has been applied to spotting and tracking human faces, and shows more robustness than other algorithms for the same task.

1 Introduction

In computer vision applications based on color-segmentation there is a common problem: the sensitivity to changes in the intensity of light. This is a result of the inability of some algorithms to tolerate variations presented in the color hue which correspond, in fact, to the same object. In this work a new algorithm based on LVQ neural networks is presented. It involves neural networks that operate directly on the image pixels with a decision function.

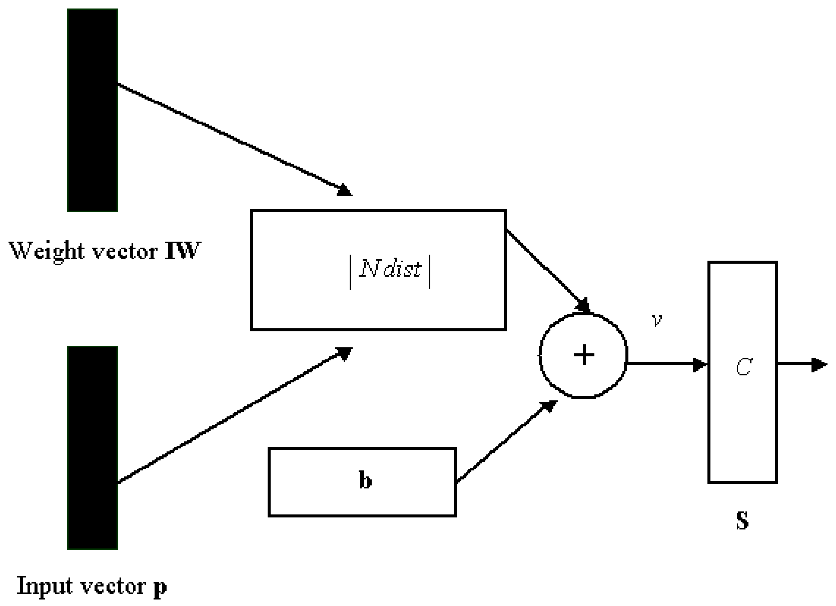


Figure 1: Architecture of a competitive network

This algorithm has been applied to spotting and tracking human faces, and shows more robustness than other algorithms for the same task.[1,2]. This work is organized in the following way: in Section 2 an introductory study of competitive neural networks and their main characteristics are presented, in Section 3 some important details of LVQ networks are analyzed, in Section 4 the architecture and characteristics of the proposed color-segmentation system are explained, in Section 5 the implementation of the algorithm is exposed, in Section 6 the Neural Network (NN) creation and training in Matlab are explained while in Section 7 the obtained results are reported.

2 Competitive Networks

Competitive Networks [3] learn to classify input vectors according to how they are grouped in the input space. They differ from another networks in that neighboring neurons learn to recognize neighboring sections of the input space. Thus, competitive layers learn both the distributions and topology of the input vectors which they are trained on.

The architecture for a competitive network is shown in Fig. 1. The $|Ndist|$ box in the figure accepts the input vector **p** and the input weight matrix **IW** and produces a vector having **S** elements. The elements are the negative of the distances between the input vector **p** and the vector **IW**. The net value v of the competitive layer is computed by finding the negative distance between input vector **p** and the weight vector **IW** and then adding the biases **b**. If, all biases are zero, the maximum net input that a neuron can have is 0. This occurs when the input vector **p** equals the neuron's weight vector.

The competitive transfer function C accepts a net value v and returns neurons outputs

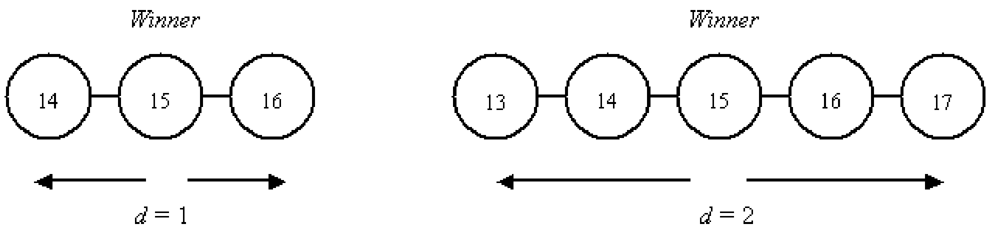


Figure 2: *Left*, one dimensional neighborhood of radius $d = 1$.*Right*, neighborhood of radius $d = 2$

of 0 for all neurons except for the *winner*, the neuron associated with the most positive element of the input v. Thus, the winner's output is 1.

The weights of the winning neuron are adjusted with the Kohonen learning rule. Supposing that the i^{th} neuron wins, the elements of the i^{th} row of the input weight matrix and all neurons within a certain neighborhood $Ni(d)$ of the winning neuron are adjusted as shown the Eq.1.

$$_i\mathbf{IW}^{1,1}(q) =_i \mathbf{IW}^{1,1}(q-1) + \alpha(\mathbf{p}(q) -_i \mathbf{IW}^{1,1}(q-1)) \tag{1}$$

Here α is the learning rate and $Ni(d)$ contains the indices for all of the neurons that lie within a radius d of the i^{th} winning neuron. Thus, when a vector $\mathbf{p}$ is presented, the weights of the winning neuron and its closest neighbors move toward $\mathbf{p}$. Consequently, after many presentations, neighboring neurons will have learned vectors similar to each others. The winning neuron's weights are altered proportional to the learning rate. The weights of neurons in its neighborhood are altered proportional to half the learning rate. In this work, the learning rate and the neighborhood distance (used to determine which neurons are in the winning neuron's neighborhood) are not altered during training.

To illustrate the concept of neighborhoods, consider the Fig. 2. At left is shown a one dimensional neighborhood of radius $d = 1$ around neuron 15, at right is shown a neighborhood of radius $d = 2$.

These neighborhoods could be written as:

$$N_{15}(1) = (14, 15, 16), N_{15}(2) = (13, 14, 15, 16, 17)$$

3 Learning Vector Quantization Networks

An LVQ network [4] has first, a competitive layer and second, a linear layer. The competitive layer learns to classify input vectors like the networks of the last section. The linear layer transforms the competitive layer's classes into target classifications defined by the user. We refer to the classes learned by the competitive layer as *subclasses* and the classes of the linear layer as *target classes*. Both the competitive and linear layers have one neuron per class. Thus, the competitive layer can learn $\mathbf{S}^1$ classes. These, in turn, are combined by the linear layer to form $\mathbf{S}^2$ target classes. The LVQ network architecture is shown in the Fig. 3.

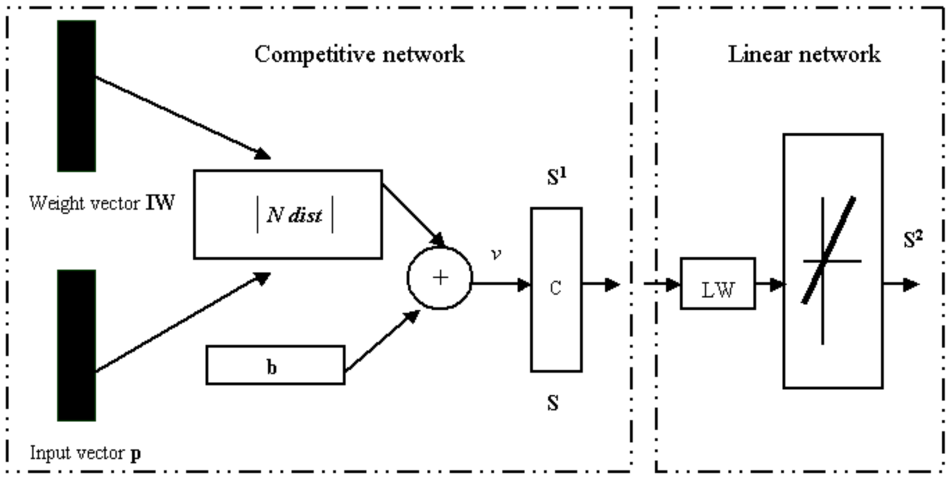


Figure 3: Schematic representation of the LVQ net

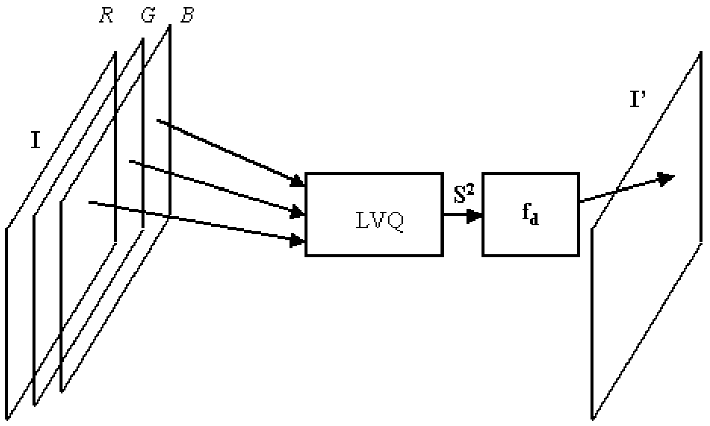


Figure 4: Architecture of the color segmentation system

Summarizing, the LVQ network allows to order classes learned by the competitive network in a final vector more appropriate for the color-segmentation.

4 Architecture of the color segmentation System

The core of the algorithm is a LVQ network whose inputs are connected directly with the pixel-vector of the image **I** (in RGB format) and its outputs are connected directly to the decision function $\mathbf{f_d}$, which produces an output of 1 or 0 depending of if the color corresponds to the object to be segmented forming in this way a new vector image **I'**. The Fig. 4 shows a scheme of the segmentator.

Considering that the LVQ net is configured with N output neurons, then it would be possible train the competitive net to learn the configuration space and the color pixels

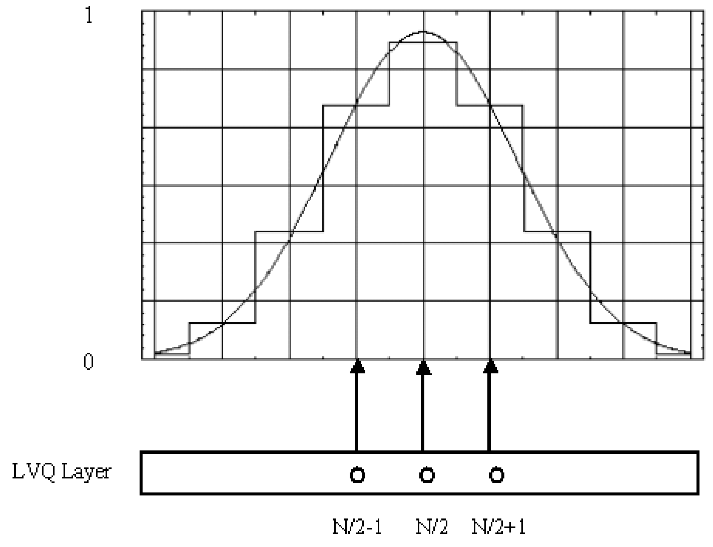


Figure 5: Decision function model

topology, described as the vector $\mathbf{p}$ with elements $\mathbf{p}_R$, $\mathbf{p}_G$ and $\mathbf{p}_B$ coming from the image. For the supervised training of the linear net, we suppose that the image $\mathbf{I}$ contains an Object O to be segmented, being $\mathbf{p_O}$ a pixel corresponding to the object, we train the linear network in such a way that the class $\mathbf{S}^2$ (of this pixel) corresponds to the assigned (in a supervised way) for the neuron N/2 of the linear network.

The idea is that the color to be segmented is located halfway of the network, while the similar colors are located in the neighboring neurons. In such a way that if exists a vector $\mathbf{p_1}$ that belongs to the object but for the illumination conditions and noise could be classified in the neighborhood of the neuron N/2.

The classification achieved by the LVQ network gives a vector of elements categorized by $\mathbf{S}^2$ of N elements corresponding to the N classes. Each element of the $\mathbf{S}^2$ vector could have 2 possible values, 1 or 0 and only an element from each vector could be 1, while the other elements will be 0. Then for the object to be segmented, the activation of the neurons is concentrated in the middle of the vector, that is the neurons nearest to the N/2 will have a bigger possibility to be activated than the other ones corresponding to the color to be segment.

Considering the previous problem is necessary to define a function $\mathbf{f_d}$ able to define neurons density which will be taken to consider if a pixel corresponds or not to the color to be segmented, this function will be called in this work decision function. It is possible to formulate many functions which could solve the decision problem satisfactorily, however the Gaussian function has been chosen by its well-known properties, although it is possible to use other, including non-simmetrical distributions. The Fig. 5 shows graphically the function used. The equation 2 shows mathematically this function where $\mathbf{g}$ is the number of the activated neuron, μ is N/2 and σ is the standard deviation. Therefore, $\mathbf{f_d}$ has only a calibration parameter represented by σ which determines the generalization capacity of the complete system. Thus, if the value of σ is chosen small enough, the segmentation capacity will be more selective than in the case of a bigger σ. For the final result the decision function was evaluated with a threshold of 0.7.

$$\mathbf{f_d}(\mathbf{g}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(- \frac{(\mathbf{g} - \mu)^2}{2\sigma^2} \right) \tag{2}$$

5 Implementation

The implementation is divided in two parts, the net training and the segmentator structure.

In the first part, a net of 30 neurons is chosen for the competitive layer and also for the linear layer, then a frame of the image is taken containing the object whose color will be segmented, then a pixel corresponding to the color of this object is chosen and the training of the LVQ net is made it, allowing that the pixel-class corresponds to the 15^{th} net class (N/2). During the training it is used a learning rate α of 0.1 and a distance radius $Ni(d)$ of 3. This conditions make sure that the winner weights are affected in a reason of 0.1 while three neurons of its neighbors (in both ways) are affected in a reason of 0.05. The process of training was made using the neural networks toolbox of Matlab.

In the second part(the segmentator implementation), a decision function with parameters μ =15 and σ =3 was added to the net (above) generated by the training. The complete system was coded in C++ and tested on a PCx86 at 900 MHz with 128 MBytes RAM, operating in real time on an image of 352x288 pixels surrendered by an USB-Webcam.

To inspect the segmentator robustness, it was tested on face localization. The segmentator gives an image with a value of 1 in those points that belong to the segmented object while in other cases this is 0. This is caused as a result that the object to be segmented in this case, will have an appearance of a white fleck. Thus, to find the object position it will be necessary to calculate the object centroid (5) through the zero moment (3) and first degree moment order (4).

$$\mathbf{M_{00}} = \sum_x \sum_y I(x,y), \tag{3}$$

$$\mathbf{M_{10}} = \sum_x \sum_y xI(x,y), \mathbf{M_{01}} = \sum_x \sum_y yI(x,y), \tag{4}$$

$$x_c = \frac{\mathbf{M_{10}}}{\mathbf{M_{00}}}, y_c = \frac{\mathbf{M_{01}}}{\mathbf{M_{00}}}, \tag{5}$$

where $\mathbf{M_{00}}$ represents the zero degree moment while of $\mathbf{M_{10}}$ and $\mathbf{M_{01}}$ means the first degree moments of $\mathbf{x}$ and $\mathbf{y}$ respectively, while $\mathbf{x_c}$ and $\mathbf{y_c}$ represent the center coordinates. The Fig. 6 shows this process.

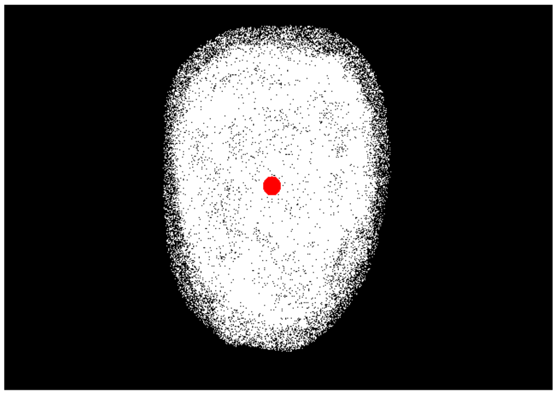


Figure 6: Center coordinates $\mathbf{x_c}$ and $\mathbf{y_c}$.

6 Neuronal Network Creation and Training in Matlab

6.1 Creating a NN in Matlab

Competitive layers and self organizing maps can be created with the Matlab commands *newc* and *newsom*, respectively. Also is possible specify different topologies for the original neuron locations with the Matlab functions *gridtop*, *hextop* or *randtop*.

In the neural networks toolbox from Matlab, there are four distinct ways to calculate distances from a particular neuron to its neighbors. Each calculation method is implemented with a special function.

In this work was used the function *dist*. It calculates the Euclidian distance from the home neuron to any other neuron.

Suppose that we want to create a network having input vectors with two elements that fall in the range 0 to 2 and 0 to 1 respectively. Further suppose that we want to have 10 neurons in a linear manner 1-by-10 network. The code to obtain this network is:

```
net=newsom([0 2; 0 1],[1,10]);
```

6.2 Training a NN in Matlab

Learning, in a self-organizing map, occurs for one vector at the time, independent of whether the network is trained directly (with the Matlab function *trainr*) or wether it is trained adaptively (with the Matlab function *trans*). In either case, *learnsom* is the self-organizing map, weight learning function in Matlab.

First, the neuron identifies the winning neuron, the weights of the winning neuron and then the neighboring neurons are moved closer to the input vector at each learning step using the self-organizing learning map (Matlab function *learnsom*). The parameters can to be modified with the values

```
LP.order_lr.
LP.order_steps.
LP.tune_nd.
```

Learning rate, Nummer of steps and neighborhood distance, respectively.

Thus, we can train the network for 100 epochs, a learning rate of 0.5 and neighborhood distance of 1. The code is:

```
net.trainParam.epochs=100;
net.trainParam.order_lr=0.05;
net.trainParam.tune_nd=1;
net=train(net,P);
```

where P is the matrix with the training patterns.

6.3 Creating and training the Linear Layer

Linear layers can be created with the Matlab command *newlin*. For example, we can create a network with two elements that be in the range -1 to 1 and one output, like is show in the followings commands:

```
net=newlin([-1 1; -1 1], 1);
```

Linear networks can be trained to perform linear classification with the Matlab function *train*. This function applies to each vector a set of input vectors and calculates the network weights and bias increments due each of the inputs, according to *learnp* Matlab function.

6.4 Auxiliary Functions

For the image capture the VFM utility was used, this Matlab-tool captures images(in RGB format) in a $m*n*3$ matrix(been mxn the dimensions of the image and the index 3 the corresponding index for R,G and B). This data matrix is represented in a integer format and it should be converted to a float point matrix(**double**). An example code for an image capture is shown.

```
vfm Im=vfm ('grab',1);
```

the captured image will be store in **Im**. **Im** should be transform for the training in a data vector of mxn type (R,G and B color channels) thats is performance by the next commands:

```
R=Im(:,:,1);
V=Im(:,:,2);
A=Im(:,:,3);
[m,n]=size(R);
index=0;
```

```
for a1=1:m
    for an=1:n
        data=R(a1,an);
        data1=V(a1,an);
        data2=A(a1,an);
        index=index+1;
        VR(index)=data;
        VV(index)=data1;
        VA(index)=data2;
        end
end
    VP=[VR;VV;VA];
    VP=double(VP);
```

Thus, from **VP** (the training vector) is possible to perform the training. The following functions sequens make that:

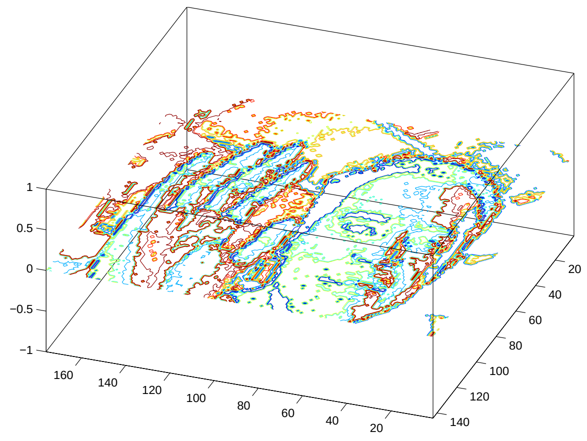
```
net=newsom([0 255;0 255;0 255],30);
net.trainParam.epochs=500;
net.trainParam.order_lr=0.1;
net.trainParam.tune_nd=3;
net=train(net,VP);
```

Ones, that the competitive neuronal network has been trained than it is trivial to find a point and their correct identification in the data vector and then make able the supervised training of the lineal network(using the Matlab function *learnp*). During the training, practically any parameters combination give good results.

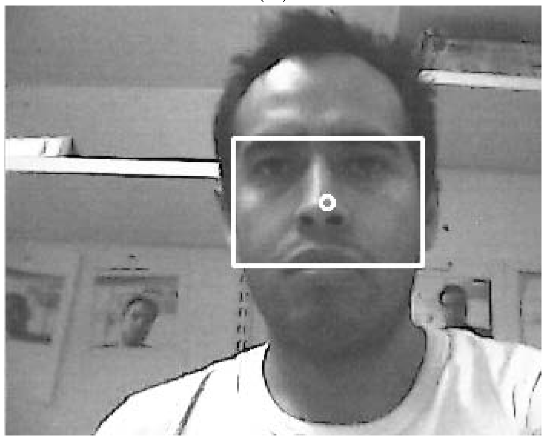
It is possible that the data obtained by the neuronal network could be interpreted as an image. To prevent this, a function was created, this function allows to modify the vector format to a mxn image format, the following commands show that:

```
index=0;
 for a1=1:m
    for an=1:n
        index=index+1;
        data=Image(index);
       MR(a1,an)=data;
    end
end
```

where Image is the data vector obtained by the neural network.



(a)



(b)

Figure 7: (a) Classification performed by a LVQ network with 10 neurons. (b) Final image of the segmentator used for face tracking

7 Conclusions.

In this section the segmentator results obtained in 3 different parts are presented.

1. In the learning of the LVQ network was proved the net classification robustness and its capacity to organize topologically, which shows excellent results, being able to perform the pixel classification of the image even in critical cases with very few neurons. The Fig. 7(a) shows the obtained classification results from an image, using a LVQ network with only 10 neurons.

2. The system was tested to track a human head showing very good results. The system performance proposed in this work compared with some other algorithms [1,2] was good with the advantage that is computationally more efficient, being better for applications where it is required, besides of the object localization, also to perform complex

calculations, for example the dynamics of a robot structure (that could supports the vision system). The Fig. 7(b). shows a tracked image in the final application.

3. The influence of the σ parameter in the decision function was demonstrate, and for this case values of σ in the interval of 3 at 5 gave good results. It can also be said that the robustness of the system can improve if it were possible to implement an algorithm to adapt the σ parameter. In the Fig. 8(a)-(b) shows the results obtained with different values of σ.

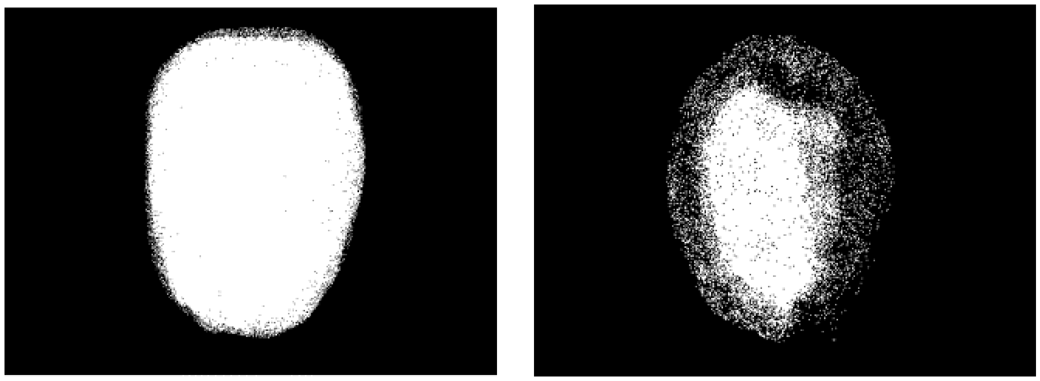

(a) (b)

Figure 8: Results considering (a) $\sigma = 4$ and (b) $\sigma = 1$.

References

[1] Jang,G.J., Kweon,I.O.: Robust Real-time Face Tracking Using Adaptive Color Model. International Conference on Robotics and Automation,(2001)

[2] Nummiaro, K., Koller-Meier, E., Van-Cool, L.: A color-based Particle Filter. Proceedings of the 1st International Workshop on Generative-Model-Based Vision,(2002)

[3] Kohonen, T.: Self-Organization and Asociative Memory. 2nd. edn., Springer-Verlag,Berlin,(1987)

[4] Kohonen, T., Self-Organizing Maps, 2nd. edn., Berlin, Springer-Verlag,Berlin,(1997)