ME6406-Machine Vision-Homework #4

Problem 1: Neural Network

a. We design our ANN as following:

-One hidden layer with Np=25 nodes which formed the layer P.

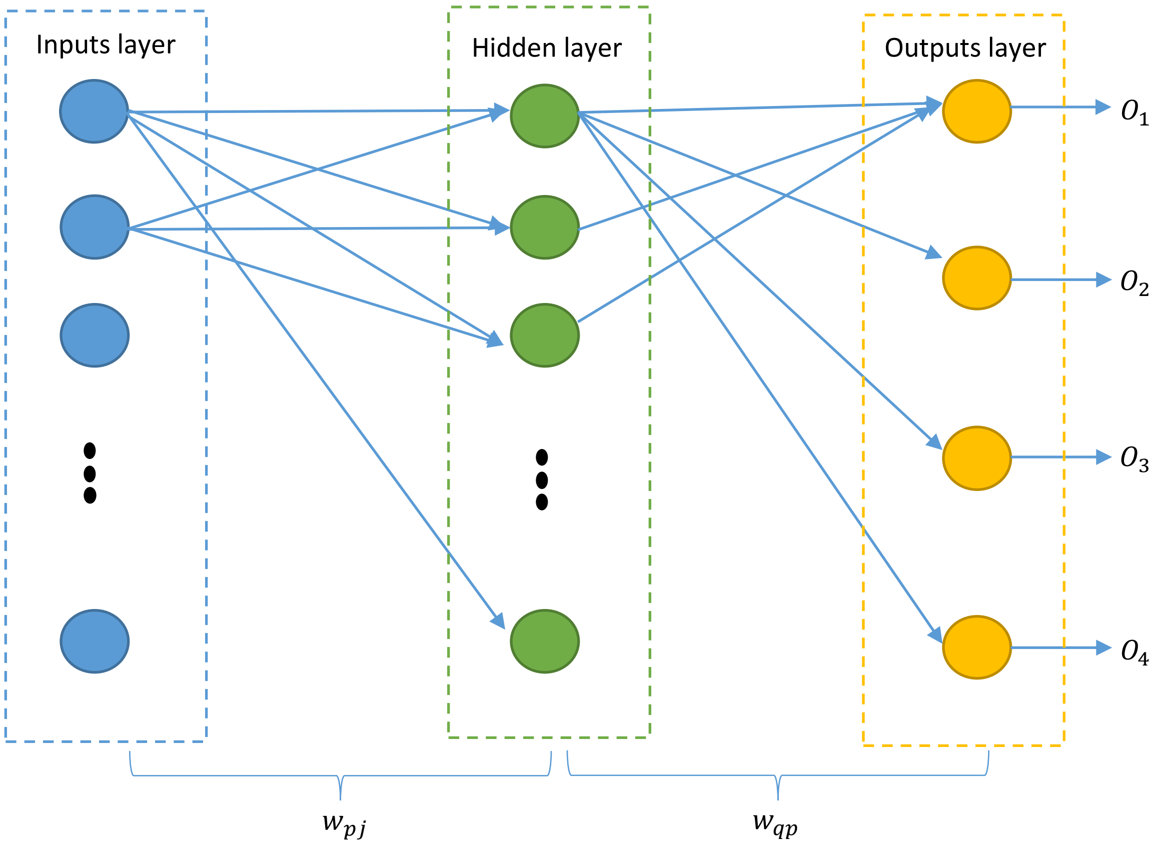
-25 inputs representing each pixels of the square grid and forming the layer J.

-4 outputs forming the layer Q.

-between two layers, each nodes of one layer are connected to each nodes of the other layer, such that there is 25x25 weights between the inputs layer and the hidden layers and there is 25x4 weights between the hidden layer and the output layer.

-the processing elements have a uni-polar sigmoid function.

We design our ANN such that we can easily modify the number of nodes of each layer in order to study the impact of the architecture of an ANN on its efficiency (learning rate, recognition accuracy & converge speed).



Derivation of the weight update rule by back-propagation learning:

$$O_i : outputs \ ; \ I_i : inputs \ ; \ r_i : references$$

Total squared error: $E_q = \frac{1}{2}\sum_{q=1}^{N_q}\left(r_q - O_q\right)^2$

Weight update rule: $w_{qp}^{k+1} = w_{qp}^{k} + \Delta w_{qp} = w_{qp}^{k} - \frac{\alpha \partial E_q}{\partial w_{qp}}$

We have to derive an expression of Δw_{qp} in order to update these weights:

$\frac{\partial E_q}{\partial w_{qp}} = \frac{\partial E_q}{\partial I_q} . \frac{\partial I_q}{\partial w_{qp}}$ where, $\frac{\partial I_q}{\partial w_{qp}} = \frac{\partial}{\partial w_{qp}}\sum_{p=1}^{N_q} w_{qp} O_p = O_p$ such that, $\Delta w_{qp} = -\frac{\alpha \partial E_q}{\partial I_q} . O_p$

Then, we pose $\delta_q = -\frac{\partial E_q}{\partial I_q}$ such that $\Delta w_{qp} = \alpha \delta_q O_p$

Moreover, we also have:

$$\frac{\partial E_q}{\partial I_q} = \frac{\partial E_q}{\partial O_q} . \frac{\partial O_q}{\partial I_q} = \frac{\partial E_q}{\partial O_q} h'(I_q)$$

Where, h is a uni-polar function, such that:

$$h(f) = \frac{1}{1+e^{-f}} \quad \Rightarrow \quad h'(f) = \frac{e^{-f}}{(1+e^{-f})^2} = \frac{1}{1+e^{-f}}\left(1 - \frac{1}{1+e^{-f}}\right) = h(f).\left(1 - h(f)\right)$$

And, we also have: $\frac{\partial E_q}{\partial O_q} = \frac{\partial}{\partial O_q}\left(\frac{1}{2}\sum_{q=1}^{N_q}\left(r_q - O_q\right)^2\right) = -(r_q - O_q)$

Finally, we get:

$$\Delta w_{qp} = \alpha\left(r_q - O_q\right)h'\left(I_q\right)O_p$$

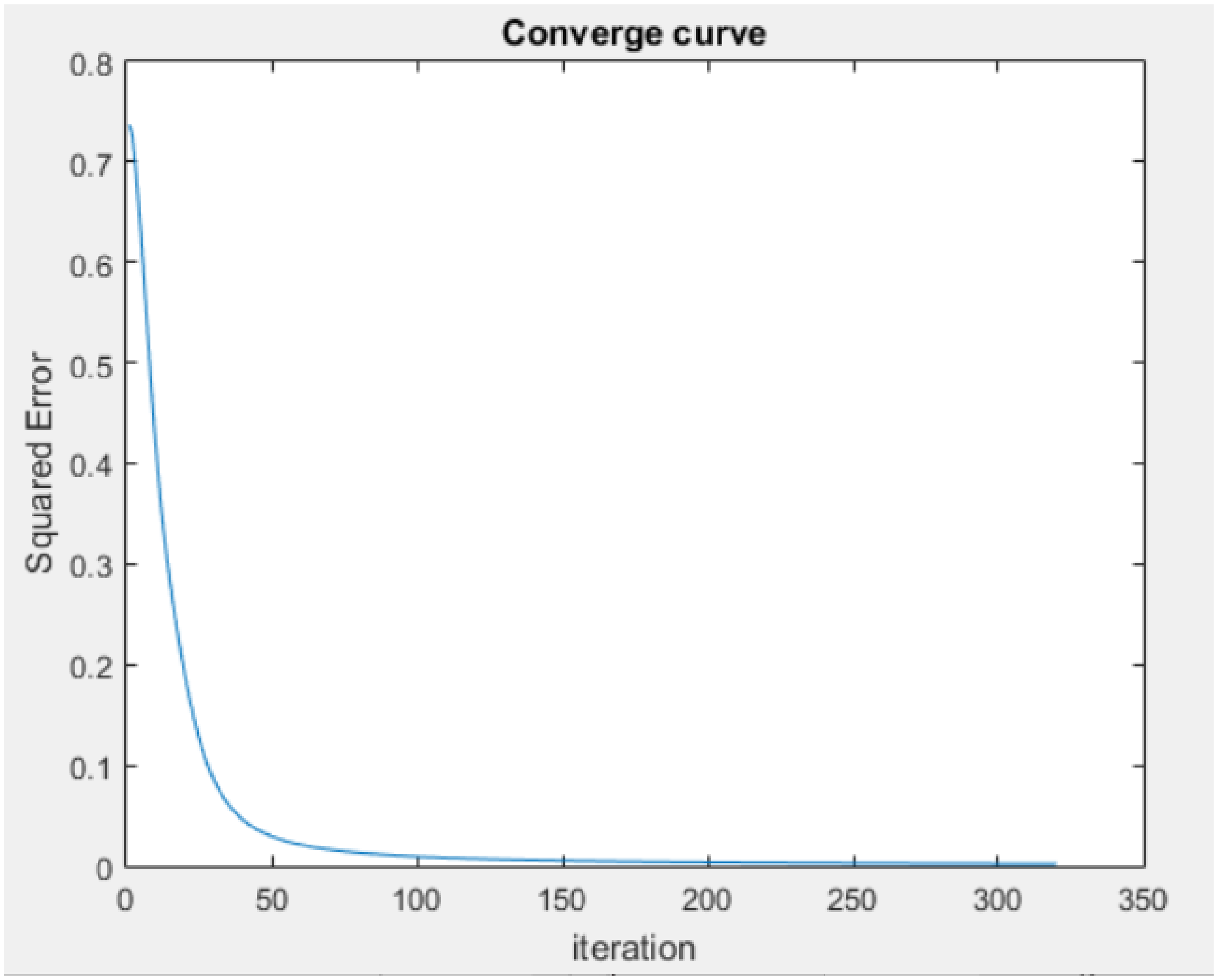
Then, we follow the same method to derive Δw_{pj} :

$\Delta w_{pj} = \alpha \delta_p O_j$ where, $\delta_p = \left(r_p - O_p\right)h'\left(I_p\right) = -\frac{\partial E}{\partial O_p} . \frac{\partial O_p}{\partial I_p} = -\frac{\partial E}{\partial O_p} . h'\left(I_p\right) = -\frac{\partial E}{\partial O_p} h\left(I_p\right)\left(1 - h\left(I_p\right)\right)$

And, $-\frac{\partial E}{\partial O_p} = -\frac{\partial}{\partial O_p}(\frac{1}{2}\sum_{q=1}^{N_q}\left(r_q - h^2\left(\sum_{p=1}^{N_q} w_{qp} O_p\right)\right) = \sum_{q=1}^{N_q}\left(r_q - O_q\right)h'\left(I_q\right). \frac{\partial I_q}{\partial O_q} = \sum_{q=1}^{N_q} \delta_q w_{qp}$

Finally, we get: $\delta_p = h'\left(I_p\right)\sum_{q=1}^{N_q} \delta_q w_{qp}$ & $\Delta w_{pj} = \alpha O_j h'\left(I_p\right)\sum_{q=1}^{N_q} \delta_q w_{qp}$

b. We write a Matlab program named NN_test.m in order to train our Neural Network. We use the function update_weight.m to update the weight with the back-propagration learning method. Then, we use the function Character_recognition.m to compute Neural Network outputs. Then, we compute the squared error by using the outputs of the NN. Our program is commented and we obtain the following converge curve of the squared error in function of the number of epoch. We train our Neural Network to get a squared error above 0.2%.



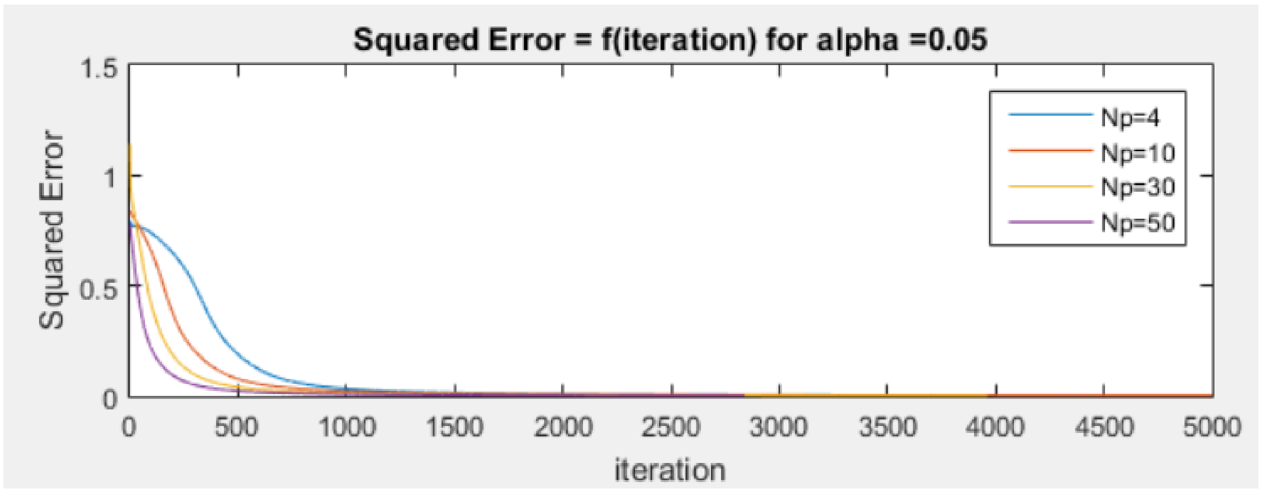
c. Then, we write a Matlab code named NN_test.m in order to test our Neural Network. We use the function Character_recognition.m to compute the outputs of the NN using the saved weights. We find the following results:

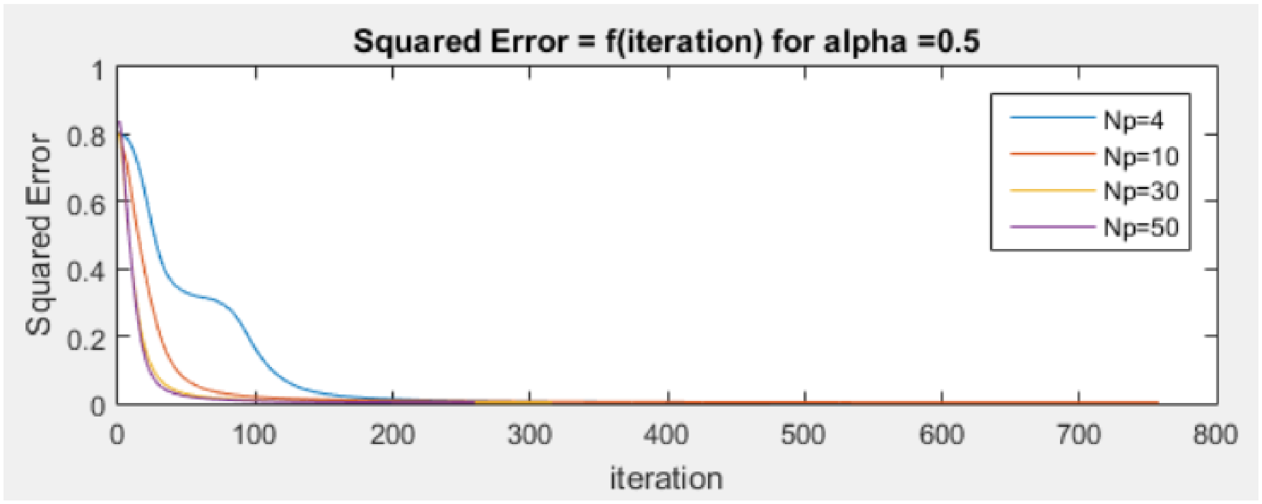
Inputs	Outputs			
G	0.95538	0.00704	0.01448	0.03893
T	0.00173	0.98227	0.00157	0.09390
M	0.02676	0.00324	0.97583	0.01029
E	0.03040	0.00554	0.00568	0.96421

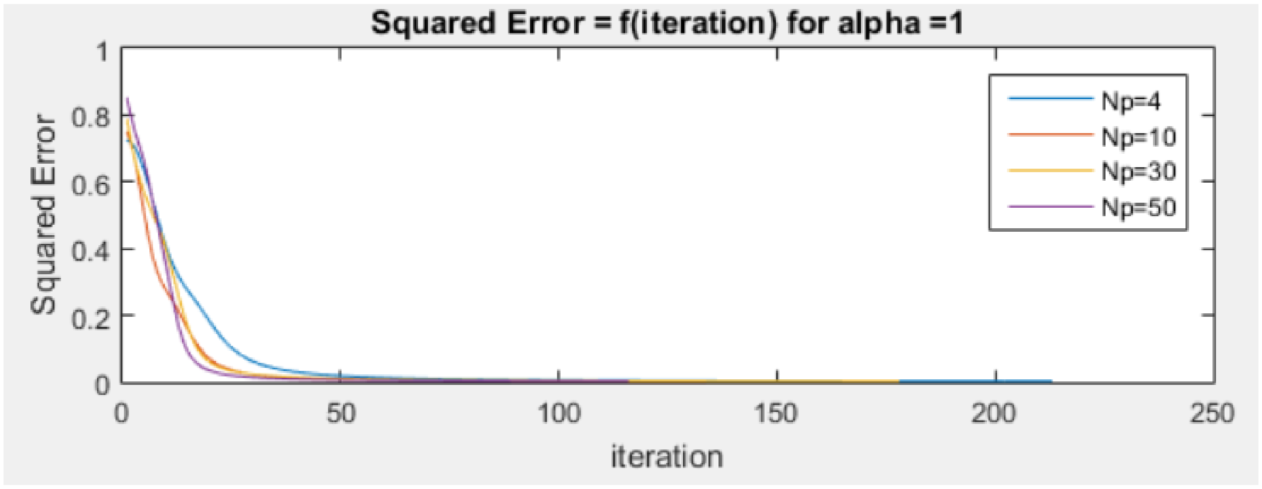
So, our NN is correctly working.

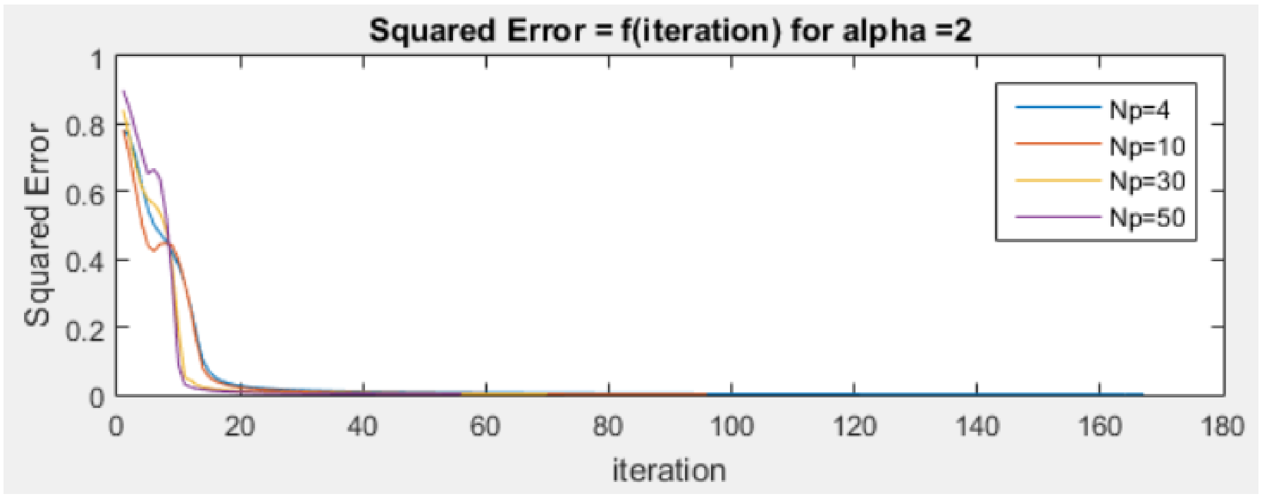
Then, we write a Matlab code named script.m which is mainly a script to study the relationship between the node numbers of the hidden layer, learning rate with the recognition accuracy and the converge speed. This script is designed to allow us to change the number of nodes in the hidden layer and the value of the learning rate, such that we can plot the squared errors for several values of Np and alpha.

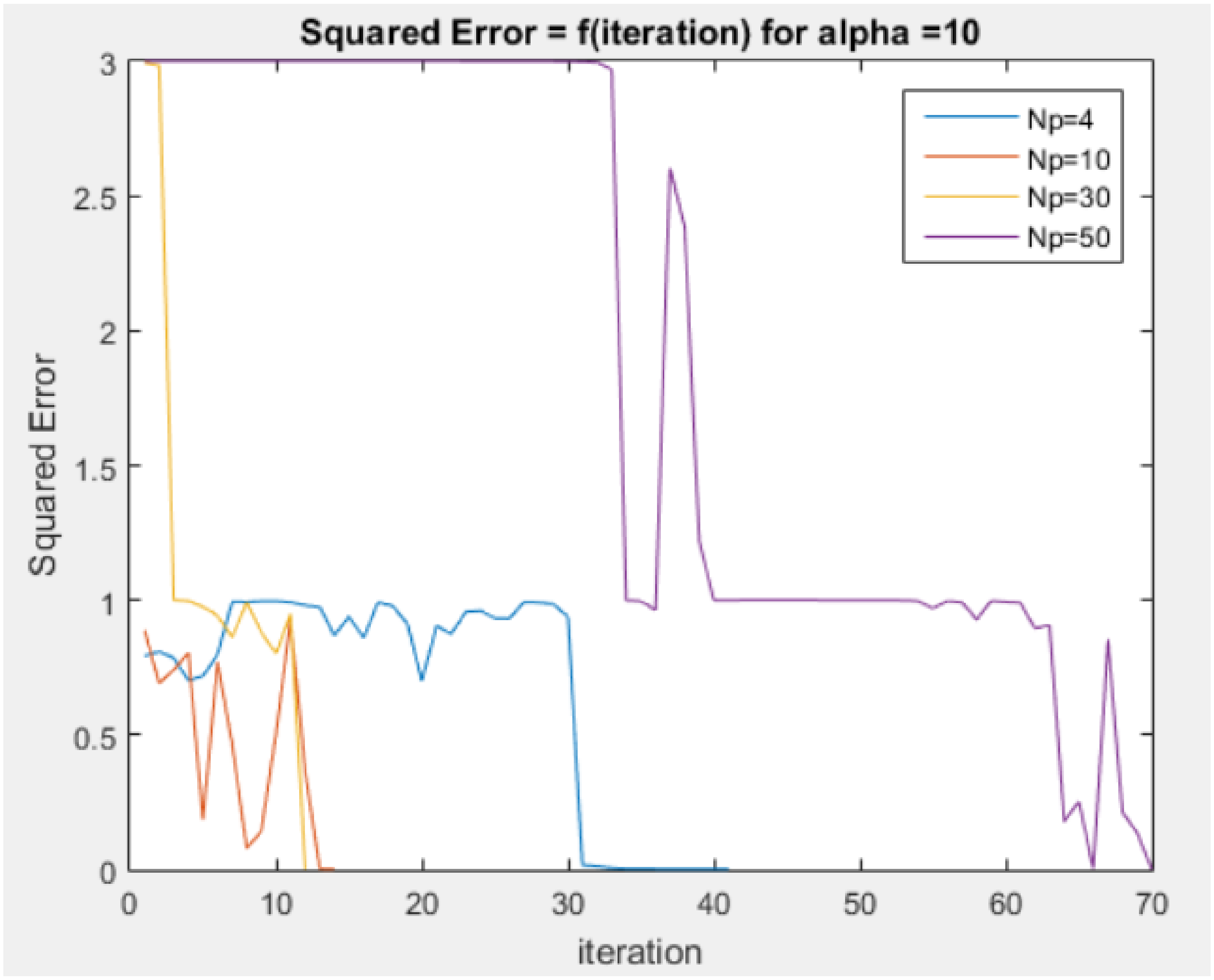
So, we plot the squared error of several values of Np for one value of alpha. We take Np equal to 4, 10, 30 and 50 for alpha equals to 0.05, 0.5, 1 and 10.




Squared Error = f(iteration) for alpha =0.5
Squared Error
iteration
Np=4
Np=10
Np=30
Np=50
0
0.2
0.4
0.6
0.8
1
0
100
200
300
400
500
600
700
800

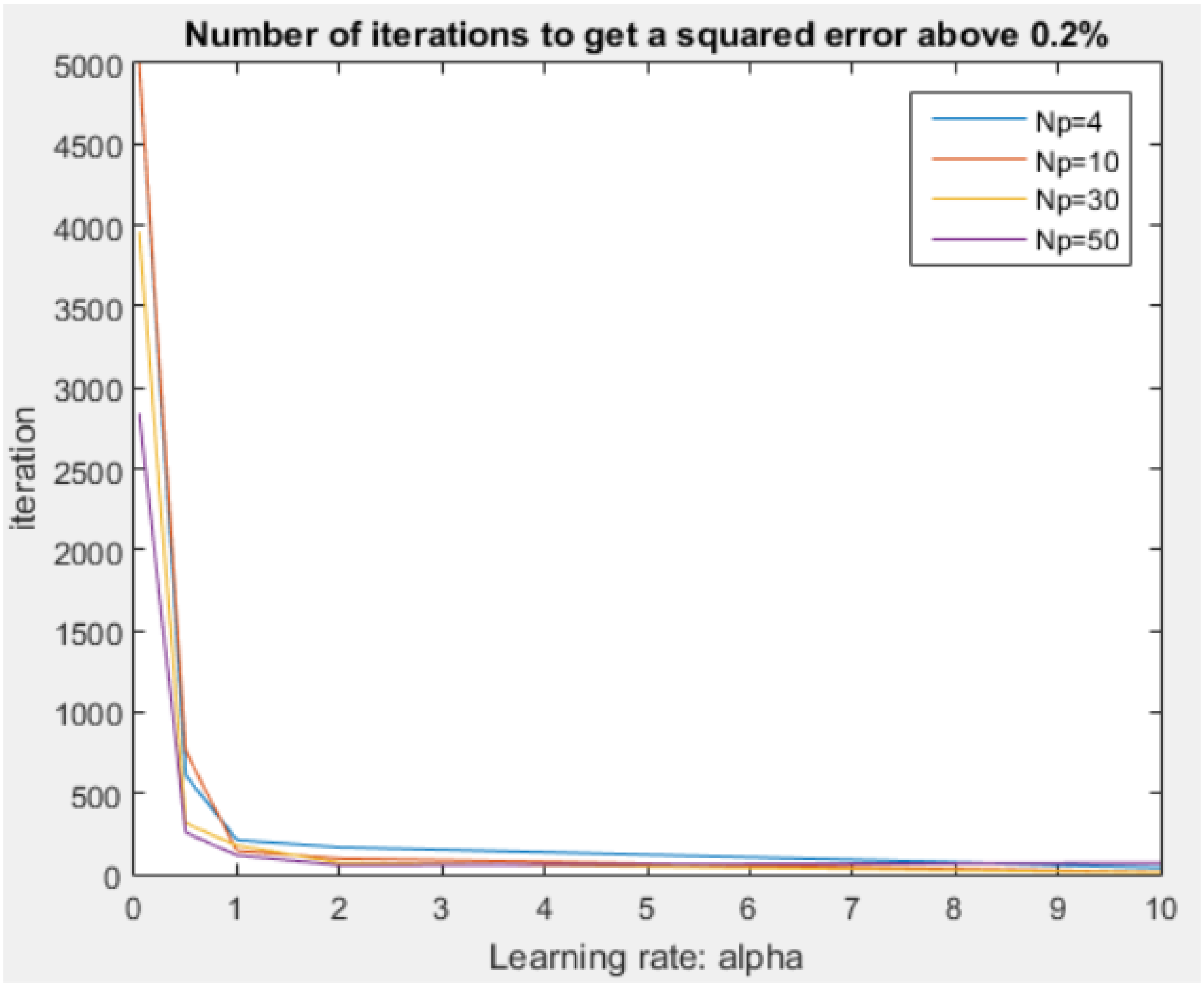

Squared Error = f(iteration) for alpha =1
Squared Error
iteration
Np=4
Np=10
Np=30
Np=50
0
0.2
0.4
0.6
0.8
1
0
50
100
150
200
250


Squared Error = f(iteration) for alpha =2
Squared Error
iteration
Np=4
Np=10
Np=30
Np=50
0
0.2
0.4
0.6
0.8
1
0
20
40
60
80
100
120
140
160
180



The first observation that we can make is that obviously the more the learning rate is high and the faster the NN will learn. That means the squared error will converge to 0 with less epochs. However, we try a high value of learning rate (alpha =10) compared to the commons used values (0.5-1) and it appears that the NN has some troubles to learn. Indeed, as we can see on the fifth plot, the squared error is not very stable and can take more epoch number to converge. This is due to the fact that even for a very small error between the outputs and the reference, the learning rate is so high that the weights changing (Δw) will not be negligible compare to the weights.

The second observation is that the more nodes we take to design the hidden layer, the fewer number of epoch the squared error needs to converge to 0 (if we take a reasonable learning rate). We can see this effect on the previous plot but also on the following plot which is the number of epoch to get a squared error above 0.2% for different number of nodes in the hidden layer in function of the learning rate.



However, increasing the number of nodes also increases the computation time. That's why, we have find a balance between accuracy, speed

Problem 2: Color

I. Artificial Color Contrast (ACC)

a.We derive the different expressions of $h_i(x,y)$ by using the following expression:

$$h_i(x,y) = G_{\sigma c} * f_j(x,y) - G_{\sigma s} * f_k(x,y)$$

For $i = 1$, we have $f_1 = R$ and we take $f_k = R - G$, such that:

$$h_1(x,y) = G_{\sigma c} * R - G_{\sigma s} * (R - G) = (G_{\sigma c} - G_{\sigma s}) * R + G_{\sigma s} * G$$

And, $DoG = G_{\sigma c} - G_{\sigma s}$

Finally, we get: $h_1(x,y) = DoG * R + G_{\sigma s} * G$

For $i = 2$, we have $f_2 = G$ and always with $f_k = R - G$, such that:

$$\begin{aligned} h_2(x,y) &= G_{\sigma c} * G - G_{\sigma s} * (R - G) = (G_{\sigma c} - G_{\sigma s}) * G + G_{\sigma s} * (2G - R) \\ &= DoG * G + G_{\sigma c} * R - G_{\sigma s} * (R - G) = (G_{\sigma c} - G_{\sigma s}) * R + G_{\sigma s} * G \end{aligned}$$

Identically, for $i = 3$ and $f_3 = B$, we get $h_3(x,y) = DoG * B + G_{\sigma s} * \big(B - (R - G)\big)$

Then, for $i = a, b, c$ we have $f_a = R$, $f_b = G$ & $f_c = B$ and we take $f_k = R + G - B$

We finally get:

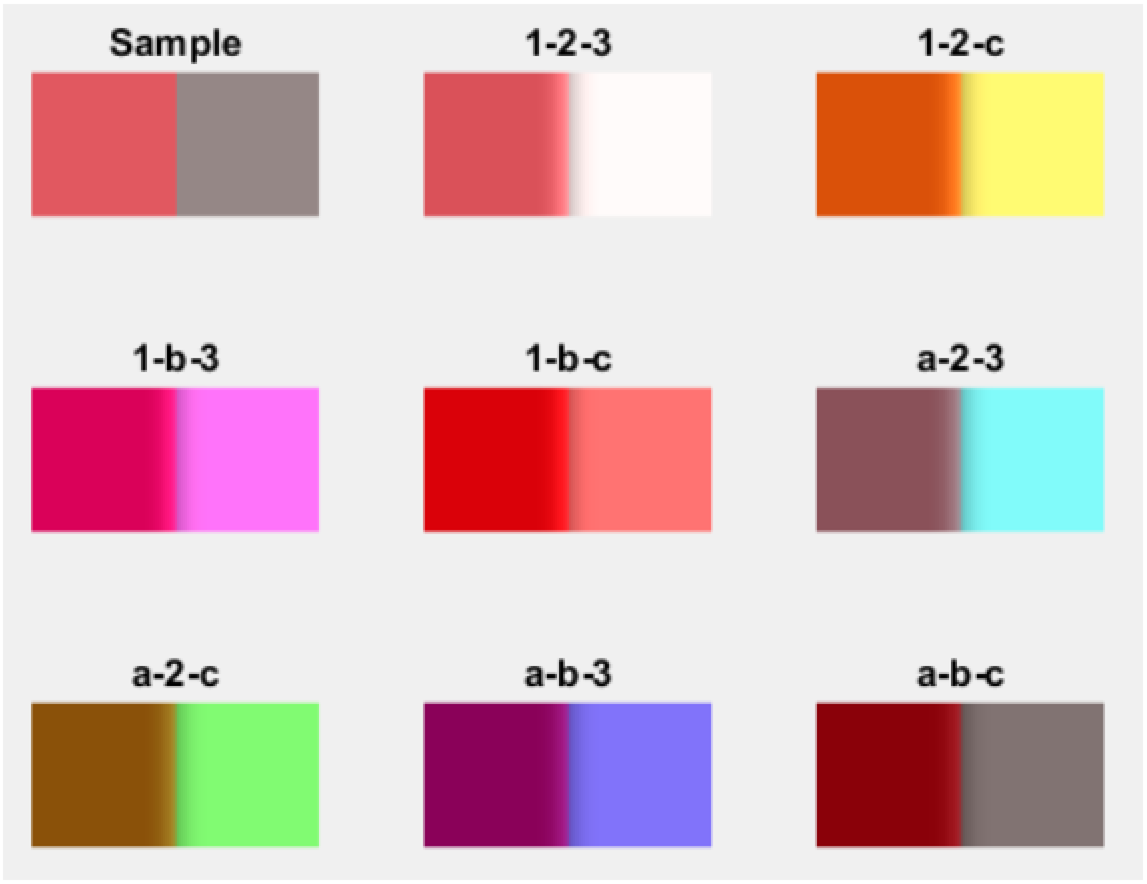
$$\begin{aligned} h_a(x,y) &= G_{\sigma c} * R - G_{\sigma s} * (R + G - B) = (G_{\sigma c} - G_{\sigma s}) * R + G_{\sigma s} * (B - G) \\ &= DoG * R + G_{\sigma s} * (B - G) \end{aligned}$$

$$\begin{aligned} h_b(x,y) &= G_{\sigma c} * G - G_{\sigma s} * (R + G - B) = (G_{\sigma c} - G_{\sigma s}) * G + G_{\sigma s} * (B - R) \\ &= DoG * G + G_{\sigma s} * (B - R) \end{aligned}$$

$$\begin{aligned} h_c(x,y) &= G_{\sigma c} * B - G_{\sigma s} * (R + G - B) = (G_{\sigma c} - G_{\sigma s}) * B + G_{\sigma s} * (2B - G - R) \\ &= DoG * B + G_{\sigma s} * (2B - (R + G)) \end{aligned}$$

b. We perform the previous transformations on two samples: [225, 88, 96] & [149, 135, 134] and we get the following results:

	Target			Noise		
	R	G	B	R	G	B
Initial	225	88	96	149	135	134
h123	88	-49	-41	135	121	120
h12c	88	-49	-121	135	121	-16
h1b3	88	-129	-41	135	-15	120
h1bc	88	-129	-121	135	-15	-16
ha23	8	-49	-41	-1	121	120
ha2c	8	-49	-121	-1	121	-16
hab3	8	-129	-41	-1	-15	120
habc	8	-129	-121	-1	-15	-16



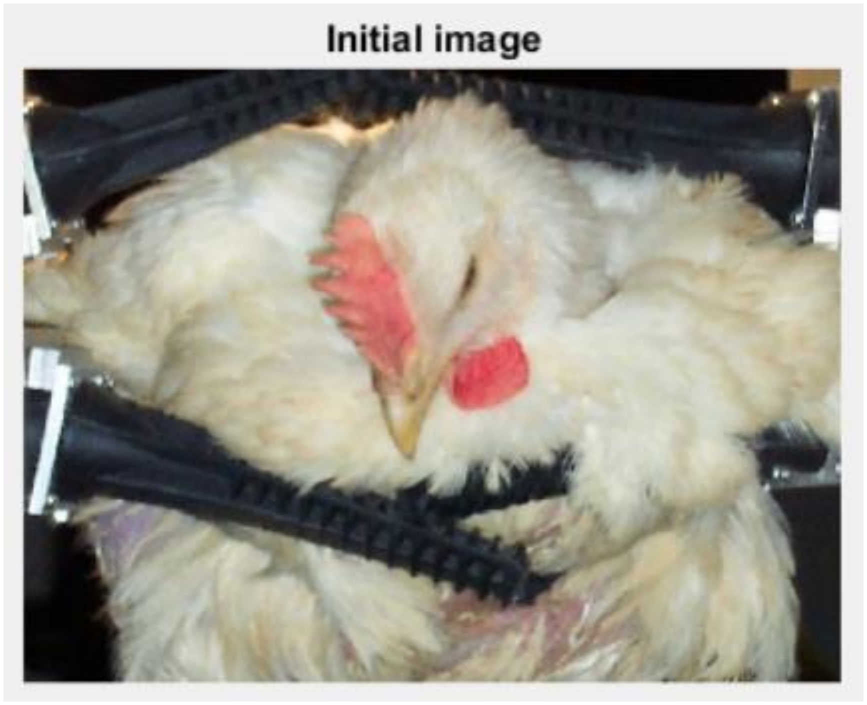
Then, we compute the distance between the target and the noise to find the maximum value. Indeed, we want to have the maximum contrast between the noise and the target in order to minimize the probability of the machine vision system to mix the noise with the target.

	h123	h12c	h1b3	h1bc	ha23	ha2c	hab3	habc	Initial
Distance Target-Noise	239	205	203	162	234	200	197	155	97

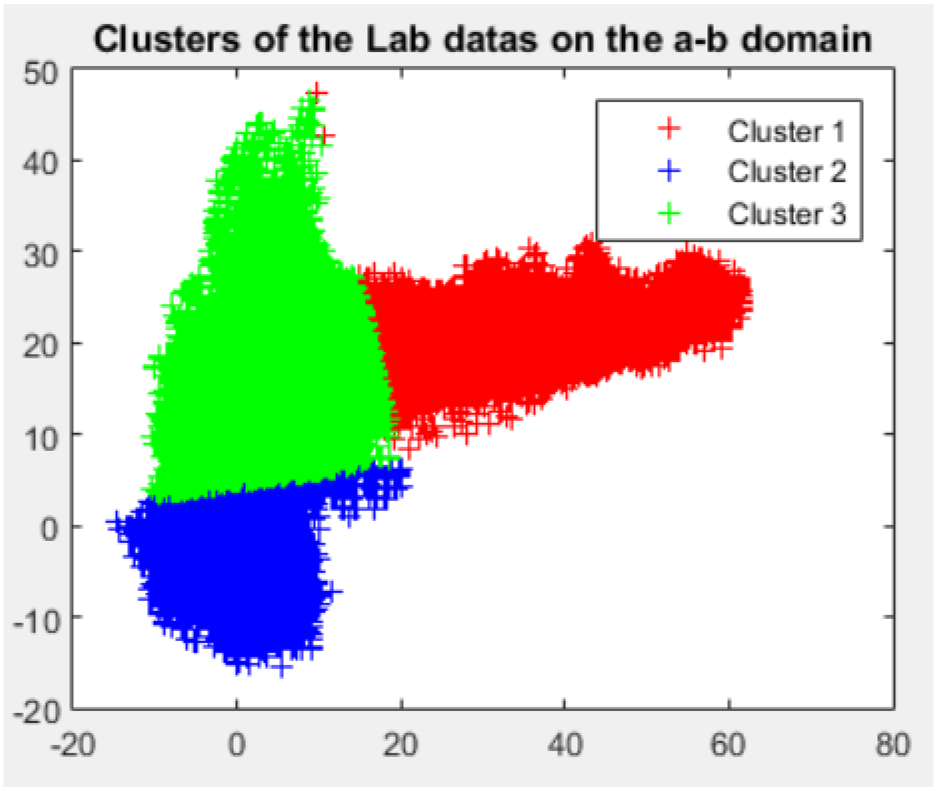
So, the best transformation is h123.

II. Color-based Image Segmentation

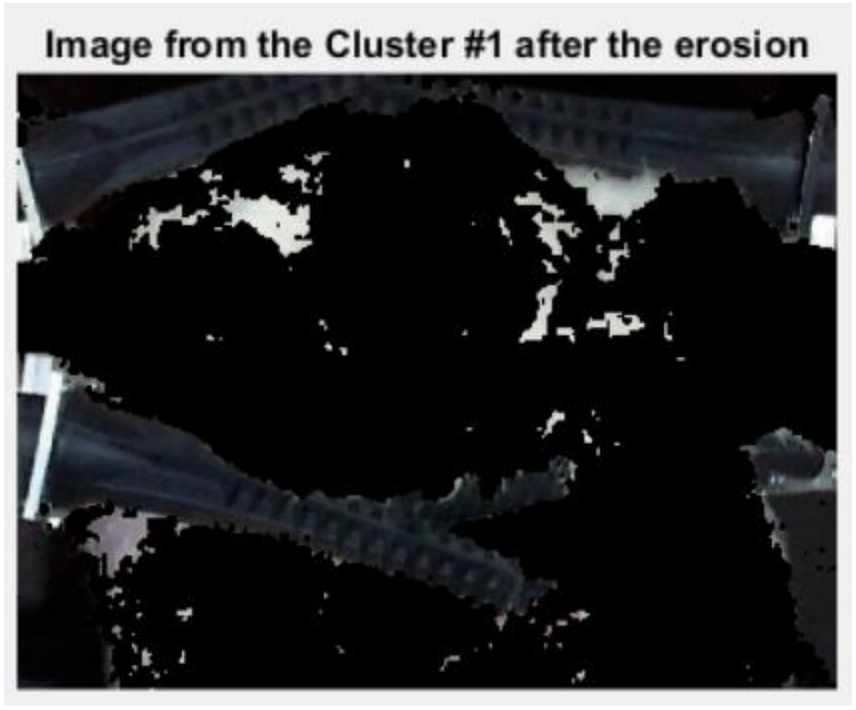
We want to perform a color-based Image segmentation on the following image in order to detect the red of the chicken head.

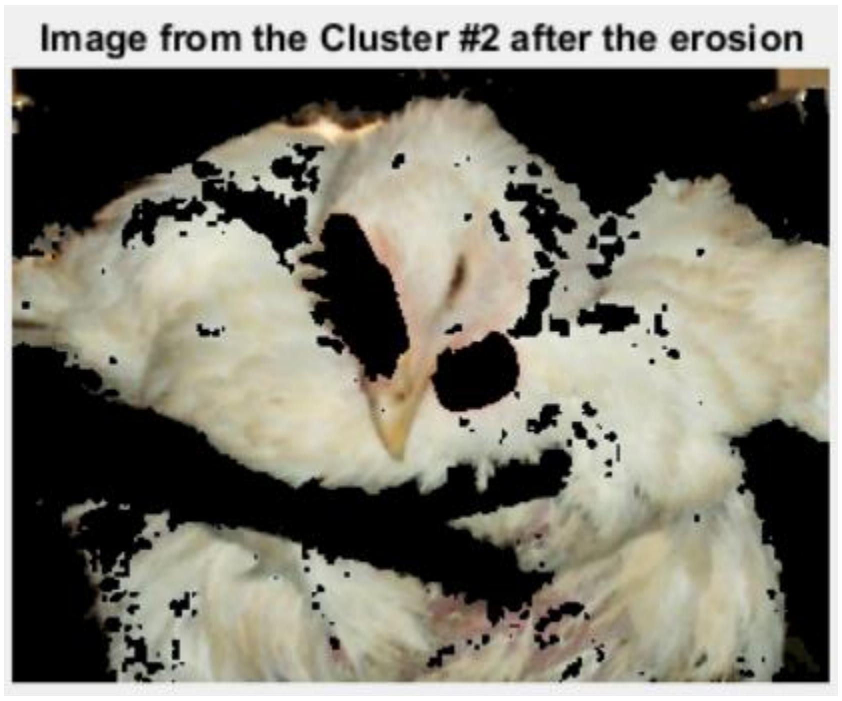


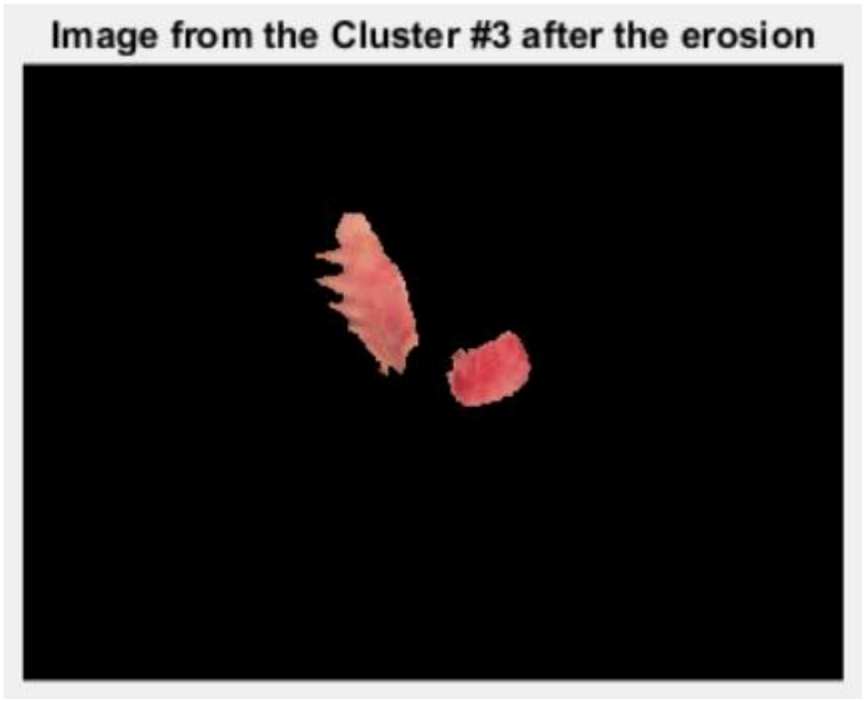
The first step is to transform the RGB image into a Lab image. We use the Matlab function rgb2lab() to get the RGB chicken image in Lab image. Then, we apply k-means clustering on the data in the a-b domain with 3 clusters and we obtain the following clustering:



The next step is to use the 3 clusters in the a-b domain to obtain three clusters in the RGB domain by using the special coordinates of the pixel. Then, we make an erosion of the three images just obtained to reduce the noise on them. Finally, you can see next these three images:





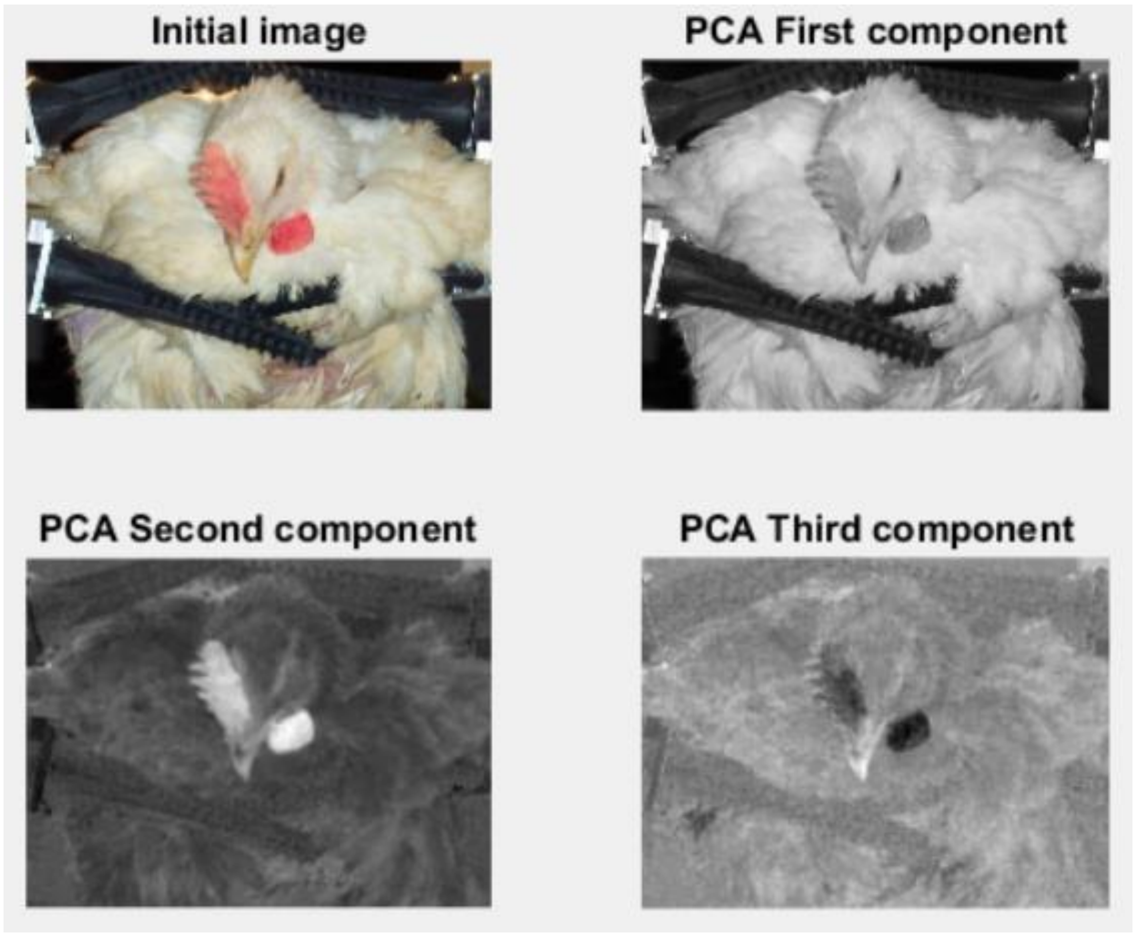


The initial image is mainly composed by three color domains. One domain near black and dark colors (background of the chicken), one near white and bright colors (the chicken) and one near the red color (crest of the chicken). So, the three clusters represent these three main color domain of the image.

So, with this method, we can easily isolate a specific domain of colors to detect a specific area as the red crest of a chicken.

III. Principle component analysis (PCA)

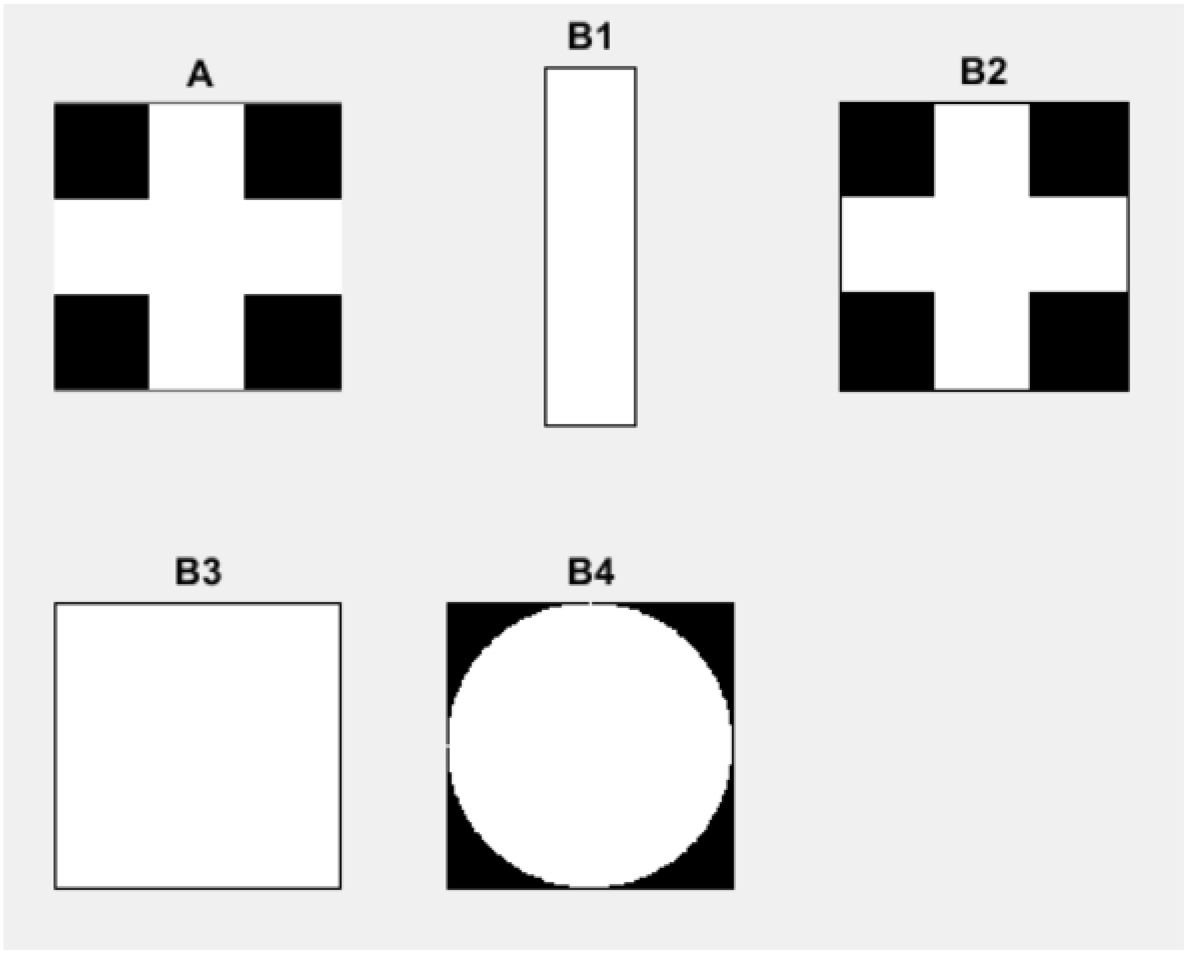
a) We use the derivation seen in class to compute the covariance of the chicken image. The equations used to compute the covariance are explained in the Matlab code.
b) Then, we use the function eig() of Matlab to get the eigenvalues/vectors and we just have to arranged these values in a descending order to get the three components in the right order.
c) Then, we scale the three components between 0 and 255 in order to be able to see the three corresponding images.



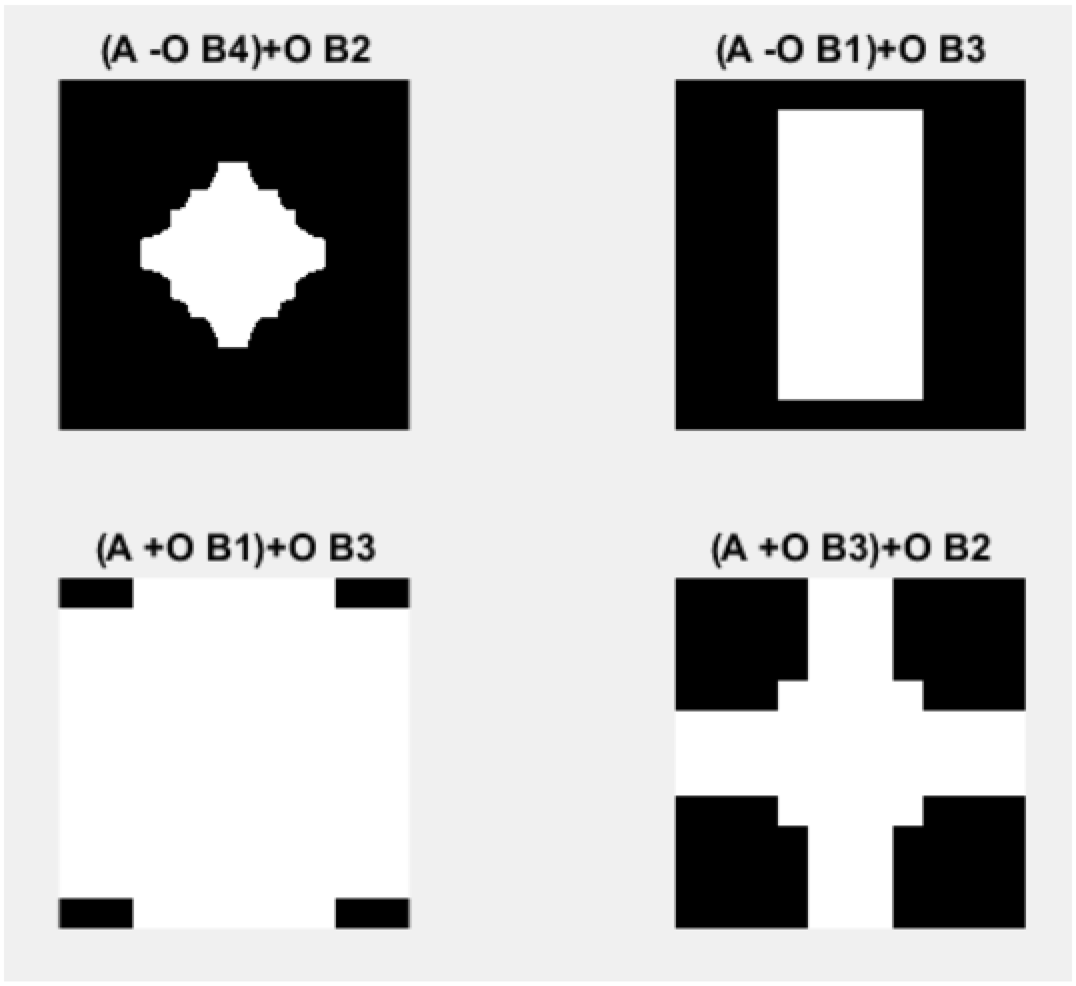
As we said previously, the principal colors of this image are black, with and red. So the first PCA component represents the black/white component, so this the contrast. Then, the second component represent the red color. Indeed, we can see that the crest which may be red is near the maximum value of this PCA component and so we see it in white. The third component seems to be the noise that is resting from the image.

Problem 3: Morphology

a) We use the following structures to study the erosion and dilatation:

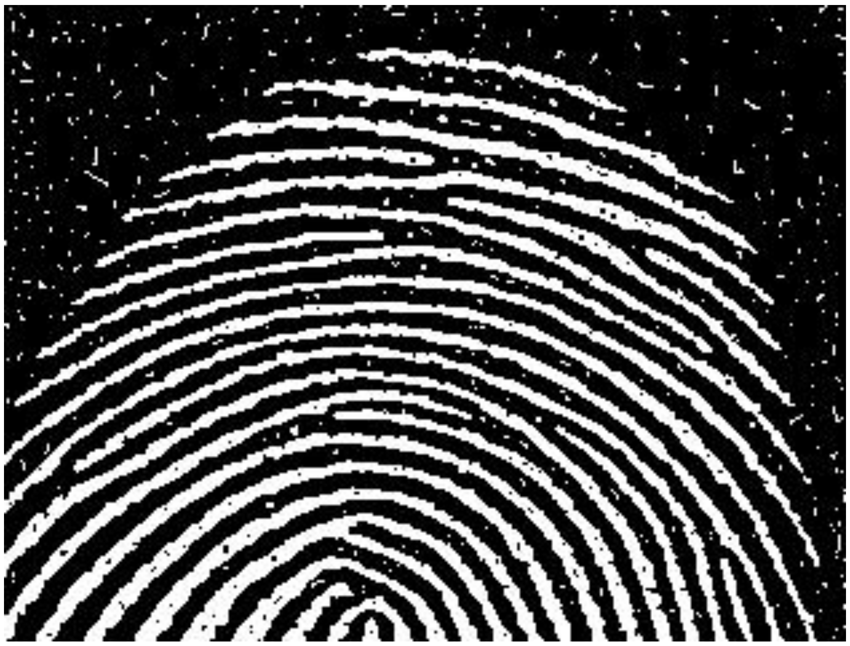


Then, we denote by "-O" the erosion and by "+O" the dilatation. We use the Matlab function imerode(A,B) for the erosion and imdilate(A,B) for the dilatation. We get the following results:



So, the dilatation is as we are moving the structure element by its middle along the boundaries of the element A and during all the way along A, we are adding the structure element to A. For the erosion, we make the same motion but instead of adding B to A, we are removing B to A.

b) We study the effect of the erosion and dilatation of a structuring element on the following fingerprint picture:



We use the following structuring element:

1	1	1
1	1	1
1	1	1

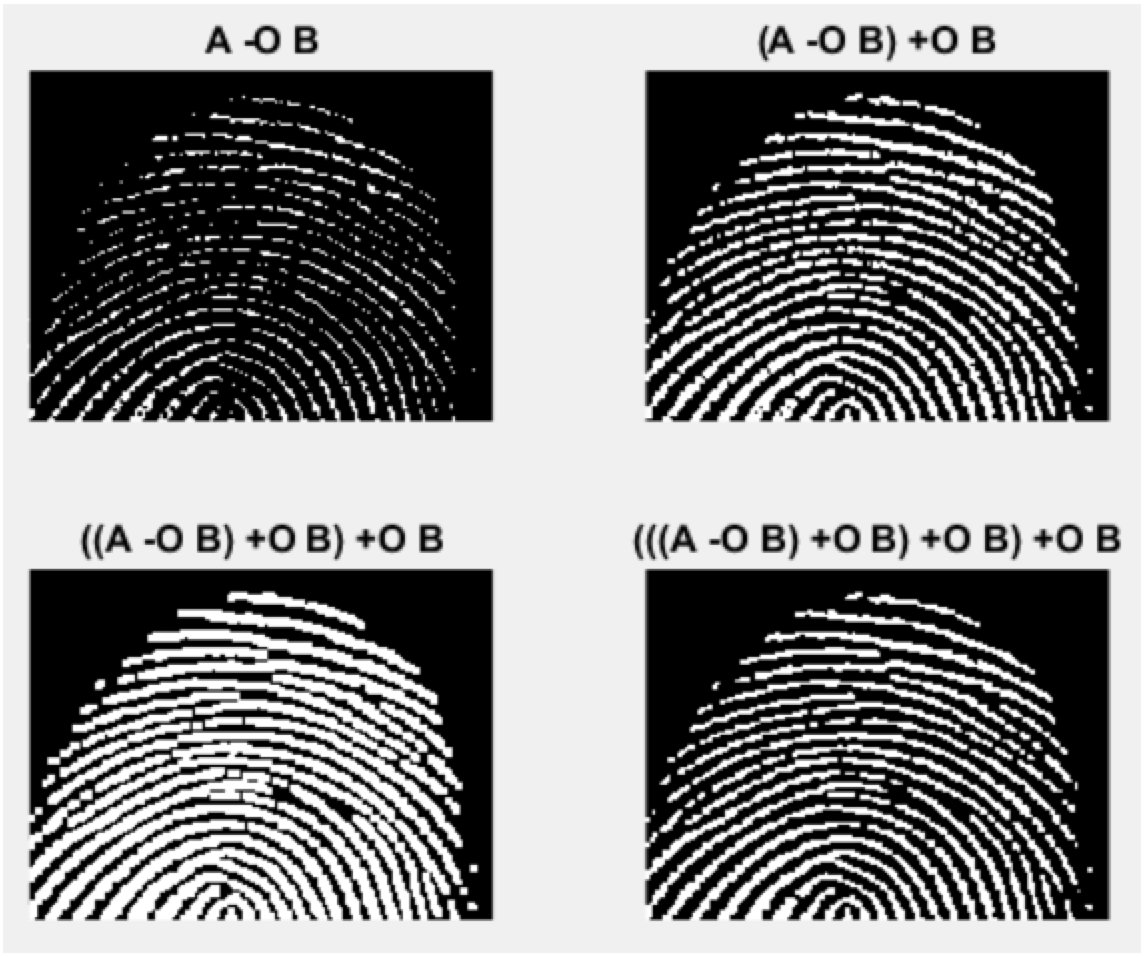
The first operation that we make is the erosion of the fingerprint by the structuring element. The result is that we delete the noise which is not on the fingerprint from the original image but we also erode the fingerprint, such that the fingerprint is thinner.

The second operation is the dilatation of the image obtained from the previous operation with the structuring element. The dilatation of the eroded fingerprint enables us to get back the thickness of the fingerprint, but without the noise around the fingerprint. So, (A –O B) +O B allows to filter the noise which is not on the fingerprint of the original fingerprint image. But, we still have the noise which is on the fingerprint.

That's why, when we dilate the previous image obtained after the two morphological operations, we get a thicker fingerprint but without noise on it. So, the fingerprint is thicker than the original image but there are no more noises.

The final step is to erode this image in order to get back the original thickness of the fingerprint without noises.

Finally, the four successive morphological operations enable us to filter the noise on this image.


A -O B
(A -O B) +O B
((A -O B) +O B) +O B
(((A -O B) +O B) +O B) +O B