**LTP function**

My LTP function returns 2 variables, ltp\_upper and ltp\_lower. The ltp\_upper variable holds the values of the upper LBP which has already been converted to uniform pattern values and the ltp\_lower variable holds the value of the lower LBP which has already been converted to uniform pattern values. The steps to do this are as follows:

Loop through the whole image from index number 2 to image size so that it doesn’t go out of bounds. We loop through each pixel and consider it as the central value. Then, we need to extract the neighbours of each pixel. So we have to extract the image from row-1:row+1 and col-1:col+1.

After extracting the neighbors we need to create the LTP pattern and store it in a temporary variable. To do this we follow the conditions. If center < center-t we have to assign the value to1. If center>center+t we need to assign the value to 1. If it is in between we need to consider the value as 0. After this we have successfully gotten the LTP values.

From the LTP pattern we need to extract the upper and lower LBP arrays from it. To extract the upper one we need to convert all the -1 in the LTP pattern as 0. To extract the lower pattern we convert all 1 to 0 and -1 to 1. After doing this we need to order it in the binary string format since right now it is in matrix 3x3 format. After making it binary string format we convert it to uniform pattern using the matlab inbuilt function bin2dec. This will finally give us the ltp\_upper and ltp\_lower matrix values.

**Pre-processing chain**

My pre-processing chain follows the same 3 steps as stated in the paper.

First, I do gamma correction with the parameter as 0.2. This will make the image more bright and clear. Since some images in the database are dark, this is very much needed.

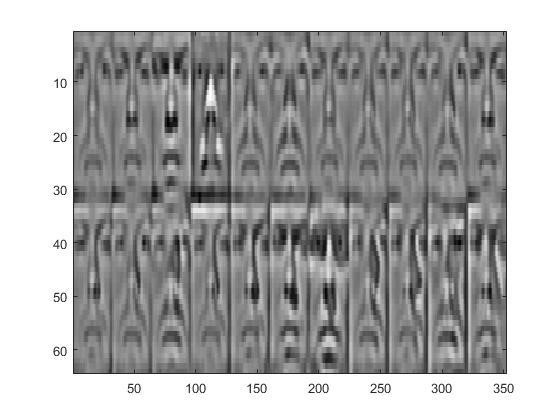
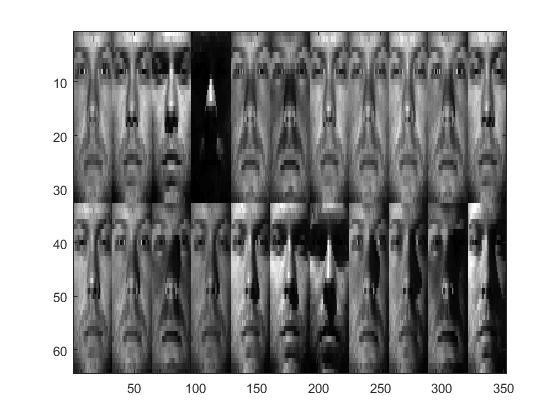
After gamma correction I apply the difference in gaussian filter with the values 1 and 2 as mentioned in the paper. This helps the image in becoming clearer as well so the features seem more well defined.

After that I perform contrast equilization again by using the formula given in the paper. I first take the average of all intensity values squared by the value a and then I factor the image by dividing it by the mean I find to the power of 1/a. I do the same in the second step except this time, I set a minimum for the intensity values which is t. so the range wont differ as much. This further helps in making the image smoother.

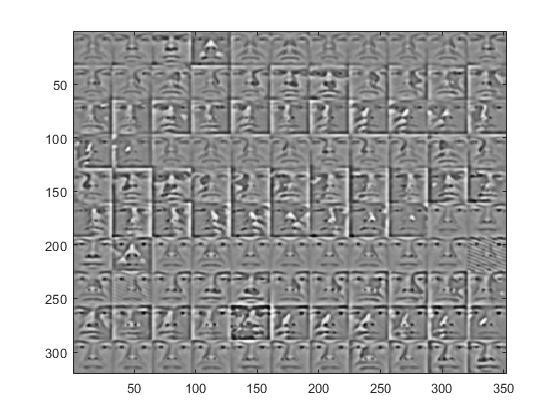
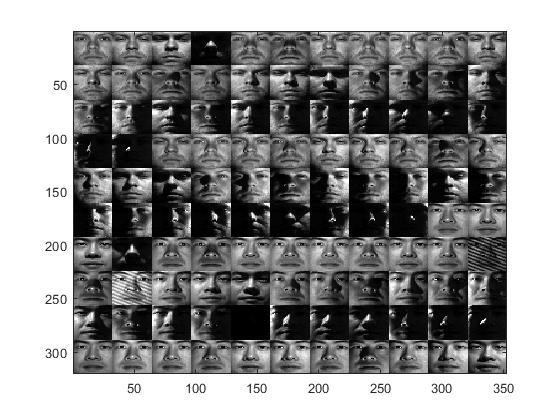
* **Examples of images**

Here are three examples of images before pre-processing and after pre-processing:

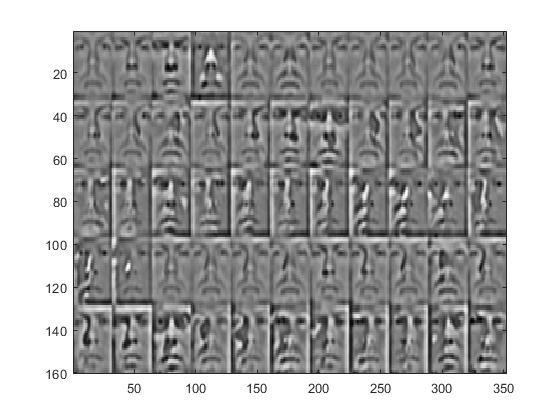
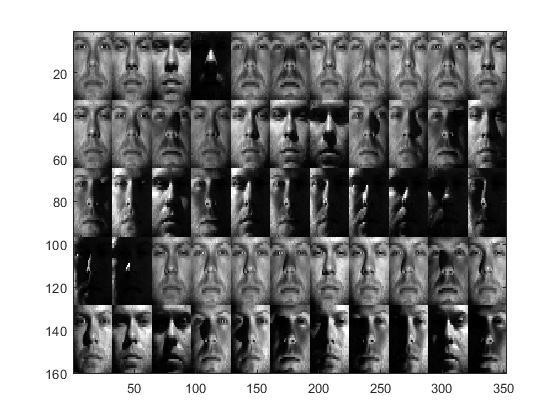
Example 1:



Example 2:



Example 3:



* **Overall recognition rates on the extended Yale-B dataset.**

The overall recognition rates are as follows:

Accuracy table:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Method | Subset 1 (401) | Subset2 (422) | Subset 3 (422) | Subset 4 (355) | Subset 5 (814) |
| PP+LTP | 77.06% | 72.98% | 70% | 75.77% | 70% |

The overall accuracy of my algorithm is 72.49% which means out of 2414 images, 1750 images get recognized correctly.

The main program calls the preprocessing for your reference and it calls two functions which is facial\_recognition.m and max\_accuracy.m to get the answer of the image input.

In the main program I loop through the whole database, and then I get the answer and compare it to the label. And if it is correct, I increase the correct count and if it is wrong I increase the wrong count. And I find the accuracy by dividing the percentage of correctly recognized images over all the images in the database.

The function calc\_similarity.m uses the equation 3 in the paper to calculate the similarity between two histograms.

The function make\_histogram.m manually creates the histogram from the LBP image. Since I didn’t know how to use any built in matlab functions for it, I just did it manually with for loops.