AANSHUL SADARIA
20161140

REPORT (DSAA) – ASSIGNMENT 1

Q1)

Nearest Neighbour Implementation:

If we want to resize an image by a factor of 'x', then we will replace each pixel by a aggregation of 'x' by 'x' pixels and assign the value of original pixel to the top left pixel of the aggregate.
Example: 255 120 => 255 _ 120 _ (Here 'x' is 2.)
_ _ _ _

The empty spaces will be filled by the nearest neighbour to them. So, as per my implementation, the resultant will matrix will look like this: 255 255 120 120
255 255 120 120

The simpler explanation to this can be that a matrix is just replaced by its 'x' by 'x' times repetition.

Hence 1 2 becomes 1 1 2 2
3 4 1 1 2 2
3 3 4 4
3 3 4 4

Bilinear Interpolation Implementation:

This one is a bit more mathematical one. As stated in the NN implementation, instead of assigning the original pixel value to the top left corner of the aggregate, I have assigned it to the bottom right without any loss of generality. Here, the empty spaces are filled such that the resized image doesn't replicate the pixels but shades (gradients) the tone.

Example: 1 _ _ _ 2 => 1 1.25 1.5 1.75 2.

Hence, the empty spaces are filled in a linear fashion and same thing is followed along both rows and columns.

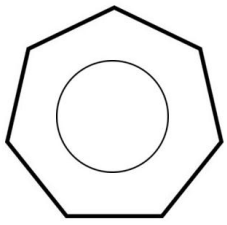
Hence 1 2 becomes -0.5 0 0.5 1.
3 4 0.5 1 1.5 2
1.5 2 2.5 3
2.5 3 3.5 4

Comparison(conclusion) and Result:

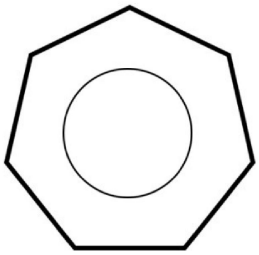
The bilinear interpolation gives much smoother output than nearest neighbour interpolation as the pixels are gradiented linearly from one original pixel to the other along both rows and columns. Bicubic interpolation gives even better results as it uses not only the weighted components of the neighbours around the pixel but also their partial derivatives. Hence, the tuning in the output image is even better.

First picture:

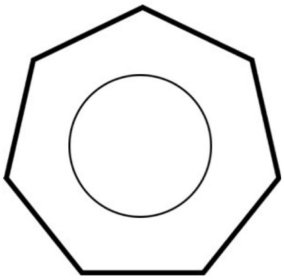
Original:



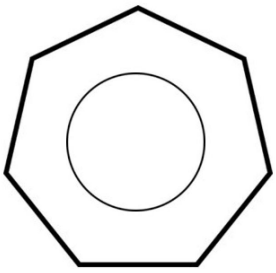
NN 5 times:



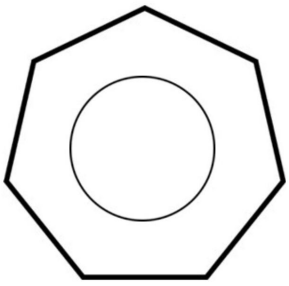
BL 5 times:



NN 2 times:



BL 2 times:



Second Picture:

Original:

NN 2 times:

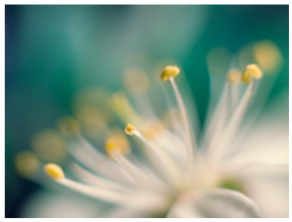
BL 2 times:

NN 5 times:

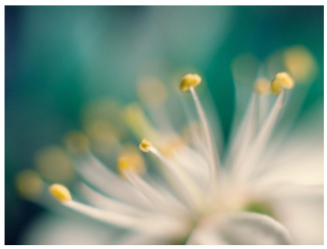
BL 5 times:

Third Picture:

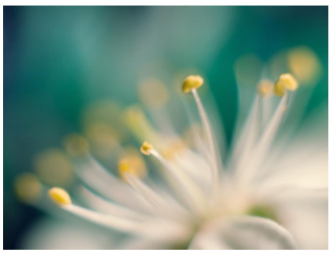
Original:



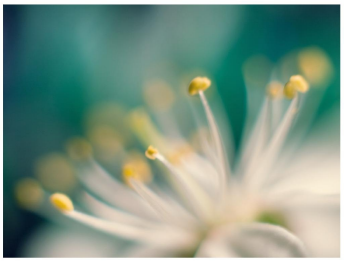
NN 2 times:



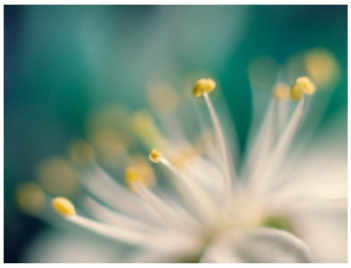
BL 2 times:



NN 5 times:



BL 5 times:



NOTE: Here, significant difference can't be seen as the image has to be resized to store in the report. Please check the code output for size confirmation.

Q2)

2D-Convolution is applied as explained in the class.
Here-in same convolution is applied, whereas the inbuilt function uses the full convolution.

a) [-1 -2 -1; 0 0 0; 1 2 1] filter is used to calculate approximately the gradient changes along the horizantal direction. It is quite similar to the edge detection filter but has '2' in the center in place of '1' and it gives smoother output.

b) Its transpose on the other hand shows gradient changes along the vertical direction.


Left: Original filter convolution
Right: Transpose filter convolution

Q3)

Resultant width (RW) = Width+2Z (padding assumed to be outside image)
Resultant height (RH) = Height+2Z
Number of convolutions along a row = (RW – F)/(S) + 1 = A
Number of convolutions along a column = (RH – F)/(S) + 1 = B
After N convolutions, convolutions along row = (Width/S^N) + (2Z-F+S)((1/S)+(1/S)^2+. . .+(1/S)^N) = X
After N convolutions, convolutions along column = (Height/S^N) + (2Z-F+S)((1/S)+(1/S)^2+. . .+(1/S)^N) = Y
Total convolutions in a channel = X*Y
a) Total dimensions = Total convolutions = X*Y*Channels (rows * columns * channels);
b)
Additions in a convolution = F*F – 1
Total additions = (F*F – 1)*(Sigma(1 to N)(X*Y))*Channels

Multiplications in a convolution = F*F
Total multiplications = (F*F)*(Sigma(1 to N)(X*Y))*Channels

Q4)

The original wave is sampled at 44100 Hz. Hence, in order to subsample it to ‘x’ Hz:

Find the ratio of change. Here it is x/44100.
Hence, from the original array we need to subsample x/44100 times the values.
The algorithm I used is as follows:

Let x/44100 be in p/q rational form.

Hence, all the elements of the original sound are repeated p times and out of the repeated array, I pick every qth element.

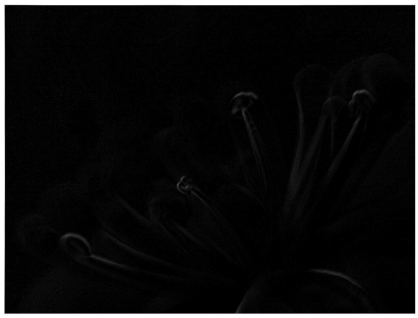
Example: [1 2 3 4]. Pick 3 values from it. 3/4 is the ratio. Hence, the repeated array is [**1** 1 1 2 **2** 2 3 3 **3** 4 4 4]. From this, pick every 4th value starting from 1st element. Resultant output will be [1 2 3].

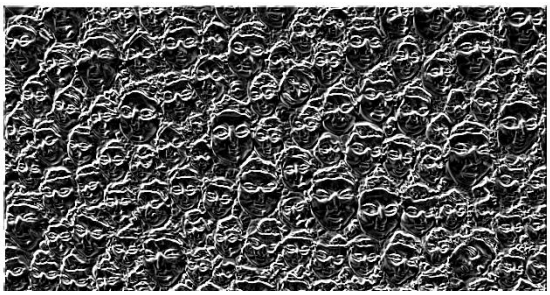
In this way, the original sound array is subsampled with lesser scope for loss of data and disorientation.

The subsampled audio can be simulated using the convolution algorithm as discussed in the class.
I observed that as I decreased the sampling rate from 44100 to 8000 the data loss increased and my voice wasn’t decipherable. Hence we have to make sure that the frequency of sampling is greater than 2f(f is the original sampling rate) in order to interpolate the signal successfully.

Q5)

The 3x3 matrix that converts as suggested is [1 1 1; 0 0 0; -1 -1 -1] which is known as the edge detection filter.
The one mentioned above is the horizontal edge detector. Whereas its transpose is the vertical edge detector filter.
Both of them give a sketch type of output (in horizontal and vertical outlines respectively).

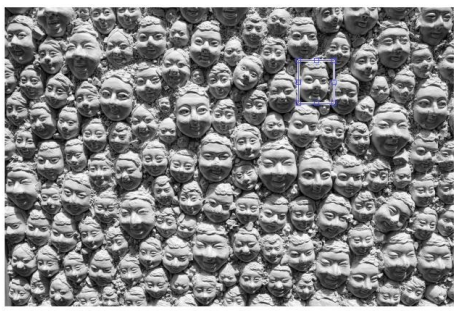




Adding the images obtained by both the filters, we get a complete outline(edges) of the input image along both the axes.

NOTE: These are the final images obtained in part(C).

Q6)



For face detection(submatrix detection), I have used the algorithm of normalized cross correlation. It is same as the full 2D-convolution, except that the filter doesn't require any flipping before applying it on the image. Also, each coefficient in the correlation matrix is divided by the root of the product of the dot product of both the matrices (filter and the superimposed part in the image) which normalizes the coefficients.

Hence, the cell of the coefficient matrix which has maximum value(1) has the submatrix coefficient. As we are using a full correlation algorithm, the cell which has the value 1

corresponds to the bottom right pixel of the cropped image. Hence, subtracting the dimensions of the cropped image from the vertex, we get the top left corner and hence we can see the cropped part in the original image as shown in the code implementation.

Q7)

Applying the same logic as explained in the class, we get 8 equations in 6 (manual calcution as it is a valid convolution) variables. Using the simple matrix operations inv(A) . X we get the values of the filter. Here, instead of 8 equations, we take only 6 and check if the result satisfies the remaining 2 equations as well.

The output (result) is as obtained in the code implementation.