CS663: Digital Image Processing

Assignment 1, Question 1

**(a) Image Shrinking**

function [OutputImage] = myShrinkImageByFactorD(InputImage, d)

% Take the file title and subsampling factor as the inputs and gives outputs the input image but subsampled by a factor of d

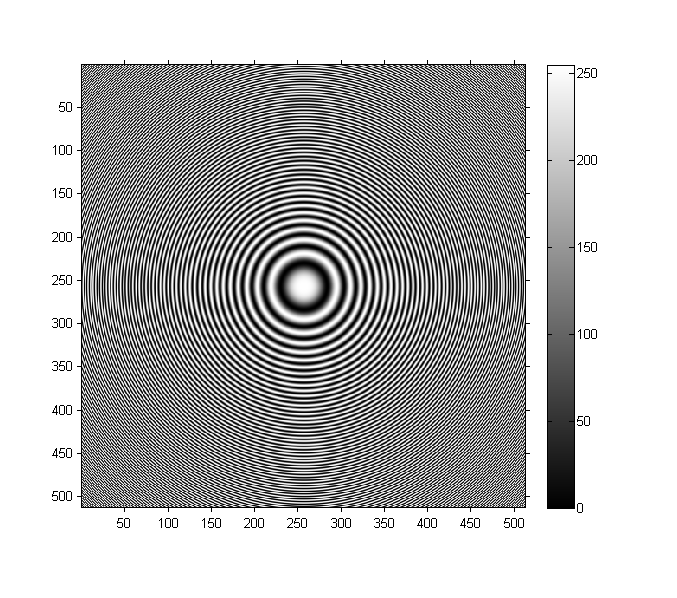
myGetBlockElement = @(X) X(1,1);

fun = @(block\_struct) myGetBlockElement(block\_struct.data);

OutputImage = blockproc(InputImage, [d, d], fun, 'UseParallel', 1);

end

**Code Explanation:** The function *myShrinkImagebyFactorD* takes *inputImage* as a single channel 8 bit image and returns *outputImage* i.e. a single channel 8 bit image with scaled dimensions. We take a block of size d X d and then take the first element of (1,1) pixel intensity of the block. These values for each d X d block are concatenated using *blockproc* to form an image of size *(sizeX / d, sizeY / d)* where *(sizeX, sizeY)* are the dimensions of the original image.

Figure. Original Image Concentric Circles

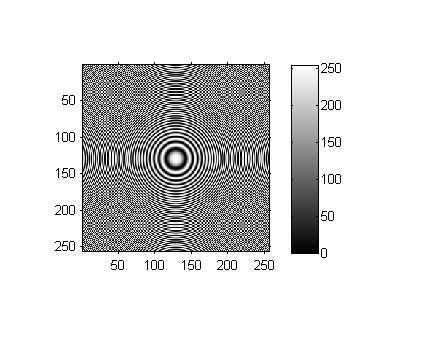


Figure. Concentric Circles, subsampled by factor of 2

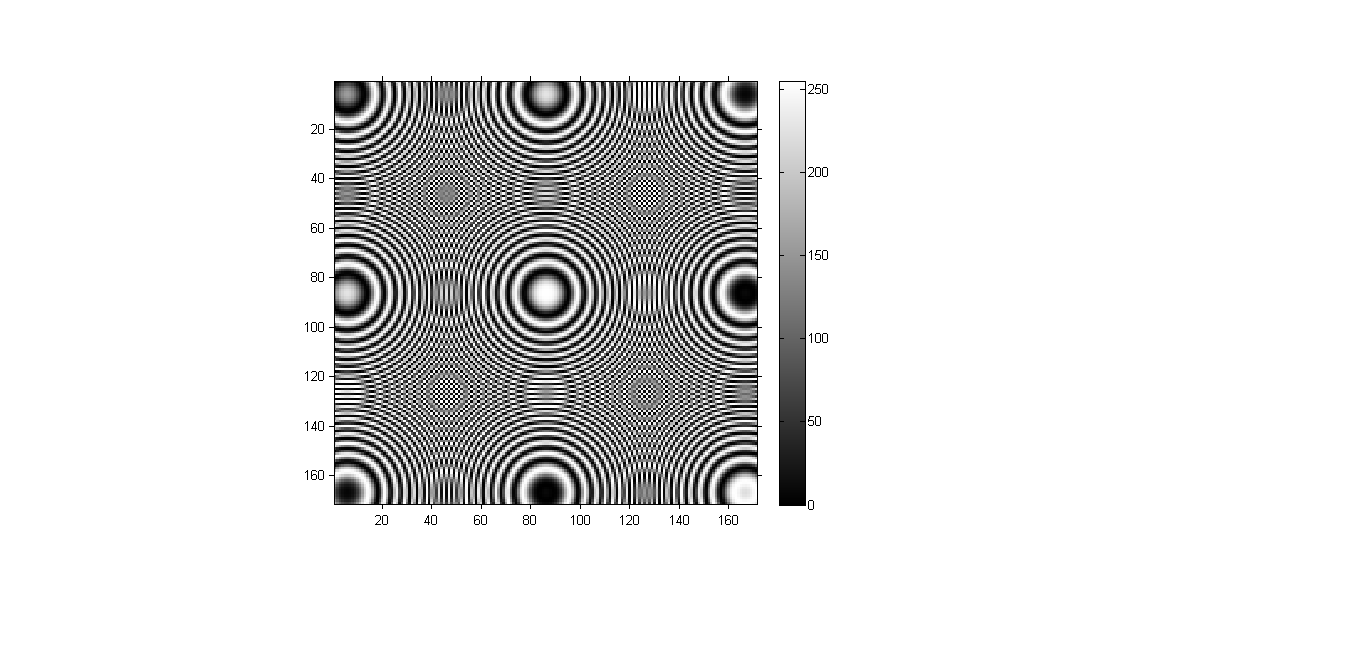


Figure. Concentric Circles, subsampled by factor of 3

**(b) Bilinear Interpolation**

function outputImage = myBilinearInterpolation(inputImage)

[R,C] = size(inputImage);

Rnew = 3\*R - 2;

Cnew = 2\*C - 1;

sR = R ./ Rnew;

sC = C ./ Cnew;

outputImage = zeros(Rnew, Cnew, 'uint8');

for rp = 2:Rnew-1

for cp = 2:Cnew-1

rF = rp .\* sR;

cF = cp .\* sC;

r = max(1, floor(rF));

c = max(1, floor(cF));

dR = rF - r;

dC = cF - c;

outputImage(rp,cp) = uint8(inputImage(r,c).\*(1-dR).\*(1-dC) + inputImage(r+1,c).\*(dR).\*(1-dC) + inputImage(r,c+1).\*(1-dR).\*(dC) + inputImage(r+1,c+1).\*(dR).\*(dC));

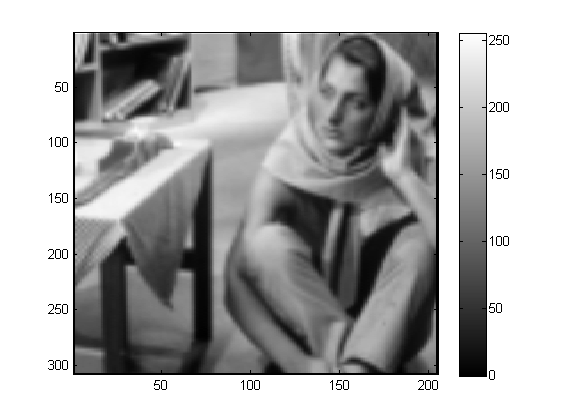
end

end

end

**Code Explanation:** The function *myBilinearInterpolation* takes *inputImage* as a single channel 8 bit image and returns *outputImage* i.e. a single channel 8 bit image with dimensions as [3R-2, 2C-1] where R and C are the rows and columns of the original image. For each pixel in the output image, intensity is calculated as linear combination of the intensities at closest adjacent points i.e. towards upper left and right corner and lower left and right corner. The indices start from 2 and go up to max index – 1 because we need to accommodate the nearby pixel values.





––

Fig. barbaraSmall before and after Linear Interpolation

**(c) Nearest Neighbour Interpolation**

function outputImage = myNearestNeighborInterpolation(inputImage)

[R,C] = size(inputImage);

Rnew = 3\*R - 2;

Cnew = 2\*C - 1;

sR = R ./ Rnew;

sC = C ./ Cnew;

outputImage = zeros(Rnew, Cnew, 'uint8');

for rp = 2:Rnew-1

for cp = 2:Cnew-1

rF = rp .\* sR;

cF = cp .\* sC;

r = max(1, floor(rF));

c = max(1, floor(cF));

outputImage(rp,cp) = uint8(inputImage(r,c));

end

end

end

**Code Explanation:** The function *myNearestNeighbourInterpolation* takes *inputImage* as a single channel 8 bit image and returns *outputImage* i.e. a single channel 8 bit image with dimensions as [3R-2, 2C-1] where R and C are the rows and columns of the original image. For each pixel in the output image, intensity is calculated as the intensity of the pixel closest to it in the original image.

