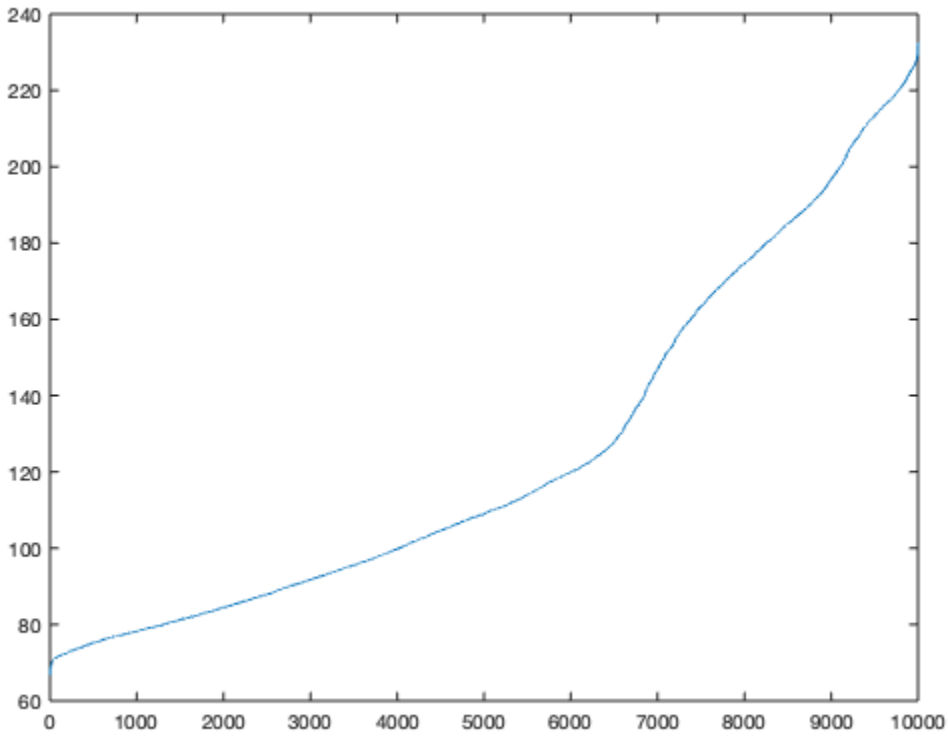
Part 0. Getting Started

```
img = double(imread('../images/flowergray.jpg'));
A = imresize(img,[100,100]);
```

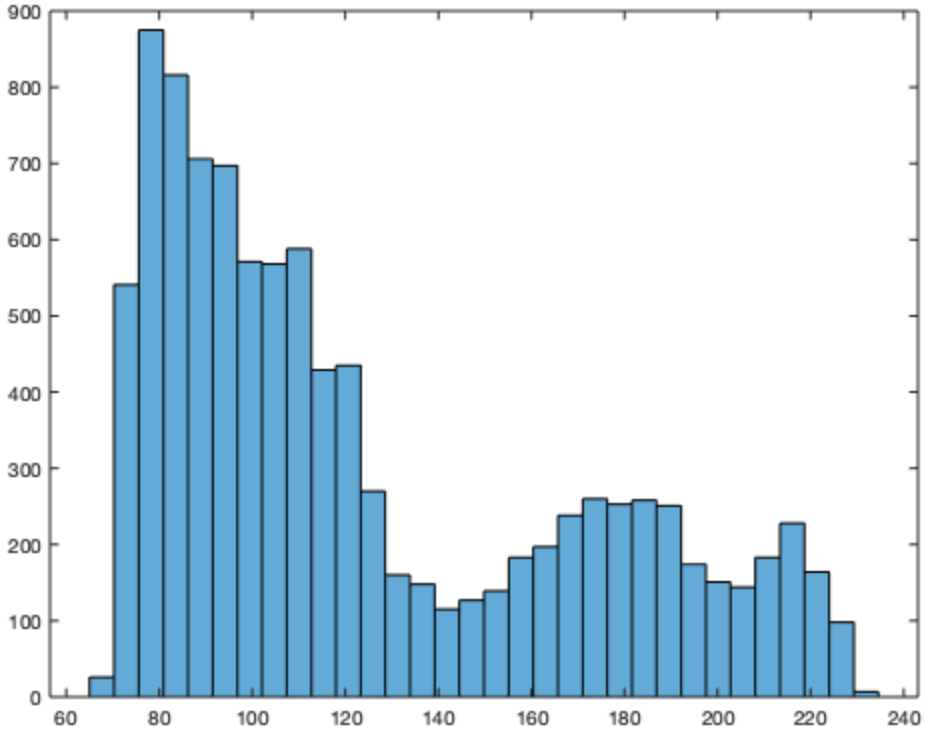
a. Sort all the intensities in A, put the result in a single 10000-dimensional vector x, and plot the values in x.

```
x = sort(A(:));
figure("Name", "Vector x");
plot(x)
```



b. Display a figure showing a histogram of A?s intensities with 32 bins.

```
figure("Name", "Histogram of A's intensities");
histogram(A,32);
```



c. Create and display a new binary image with the same size as A, which is white wherever the intensity in A is greater than a threshold t, and black everywhere else.

```
t = 100;
binaryImage = imbinarize(A, t);
figure("Name", "Binary image A");
imshow(binaryImage)
```

d. Generate a new image (matrix), which has the same size as A, but with A?s mean intensity value subtracted from each pixel. Set any negative values to 0.

```
meanIntensity = mean(mean(A));
Aprime = A - meanIntensity;
Aprime(Aprime < 0) = 0;
```

e. Let y be the vector: y = [1: 8]. Use the reshape command to form a new matrix s whose first column is [1, 2, 3, 4]?, and whose second column is [5, 6, 7, 8]?.

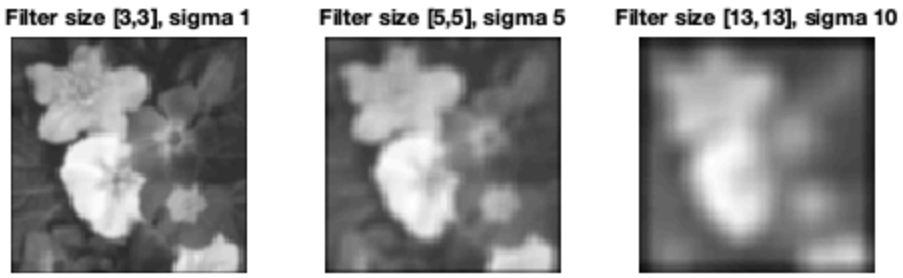
```
y = [1:8];
newY = reshape(y,[4,2]);
```

f. Create a vector [1, 3, 5 ?, 99]. Extract the corresponding pixel from theimage in its two dimensions, i.e., subsample the original image to its half size.

```
indexs = (1:2:99);
downImage = A(indexs,indexs);
```

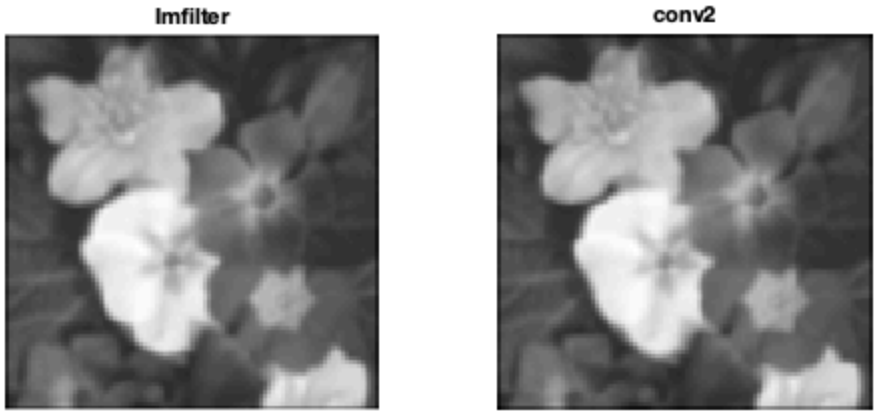
g. Use fspecial to create a Gaussian Filter and then apply the imfilter function to the image with the created Gaussian Filter, by doing so you should see a blurred image. Change three combinations of parameters of the Gaussian Filter and compare the results.

```
filter1 = fspecial('gaussian',[3,3], 1);
image1 =  imfilter(A, filter1);
filter2 = fspecial('gaussian',[5,5], 10);
image2 =  imfilter(A, filter2);
filter3 = fspecial('gaussian',[13,13], 10);
image3 =  imfilter(A, filter3);

figure("Name", "Gaussian filter");

subplot(131)
imshow(image1,[])
title("Filter size [3,3], sigma 1");

subplot(132)
imshow(image2,[])
title("Filter size [5,5], sigma 5");

subplot(133)
imshow(image3,[])
title("Filter size [13,13], sigma 10");
```



h. Apply the conv2 instead of imfilter function to the same process (for one Gaussian Filter), do you see any changes? Why?

```
conImage1 = conv2(A, filter1, 'same');

figure("Name", "Conv vs imfilter");

subplot(121)
imshow(image1,[])
title("Imfilter");

subplot(122)
imshow(conImage1,[])
title("conv2");
% *No changes, they are same*
```



Published with MATLAB® R2019b

Part 1: Gaussian Pyramid

```
images = ["../images/CARTOON.jpg", "../images/flowergray.jpg", "../
images/kitty.jpg", ...
    "../images/polarcities.jpg", "../images/text.jpg" ];

filter = [0.25,0.25; 0.25,0.25];

for image = images

    img = im2double(imread(image));
    [height, width] = size(img);

    n = log2(height);

    figure;
    subplot(3, 3, 1);
    imshow(img);
    title('Original image')

    for i= 1:n
        img = imfilter(img, filter, 'replicate', 'same');
        [height, width] = size(img);

        % Half size
        img = img(1:2:height, 1:2:width);

        % Produce same size of original image
        biImage = imresize(img, 2^i, 'bilinear');

        subplot(3, 3, 1+i);
        imshow(biImage);
        title(sprintf('%d times filtering', 2 ^ i));
    end
end
```

Original image

2 times filtering

4 times filtering

8 times filtering

16 times filtering

32 times filtering

64 times filtering

128 times filtering

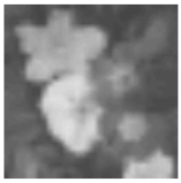
256 times filtering

Original image

2 times filtering

4 times filtering

8 times filtering



16 times filtering

32 times filtering

64 times filtering

128 times filtering

256 times filtering

Part 2: Laplacian Pyramid

```
images = ["../images/CARTOON.jpg", "../images/flowergray.jpg", "../
images/kitty.jpg", ...
    "../images/polarcities.jpg", "../images/text.jpg" ];

filter = [0.25,0.25; 0.25,0.25];

for image = images

    img = imread(image);
    [height, width] = size(img);
    side = log2(height);

    figure;

    for i= 1:side
        filteredImg = imfilter(img, filter, 'replicate', 'same');
        [height, width] = size(filteredImg);

        % Half size
        filteredImg = filteredImg(1:2:height, 1:2:width);

        % Produce same size of original image
        biImage = imresize(filteredImg, 2, 'bilinear');

        % Laplacian pyramid
        pyramid = biImage - img;

        img = filteredImg;
        subplot(4, 2, i);
        imshow(pyramid);
        title(sprintf('Laplacian Pyramid %d', i));
    end
end
```

Laplacian Pyramid 1

Laplacian Pyramid 2

Laplacian Pyramid 3

Laplacian Pyramid 4

Laplacian Pyramid 5

Laplacian Pyramid 6

Laplacian Pyramid 7

Laplacian Pyramid 8

Laplacian Pyramid 1

Laplacian Pyramid 2

Laplacian Pyramid 3

Laplacian Pyramid 4

Laplacian Pyramid 5

Laplacian Pyramid 6

Laplacian Pyramid 7

Laplacian Pyramid 8

Laplacian Pyramid 1

Laplacian Pyramid 2

Laplacian Pyramid 3

Laplacian Pyramid 4

Laplacian Pyramid 5

Laplacian Pyramid 6

Laplacian Pyramid 7

Laplacian Pyramid 8

Laplacian Pyramid 1

Laplacian Pyramid 2

Laplacian Pyramid 3

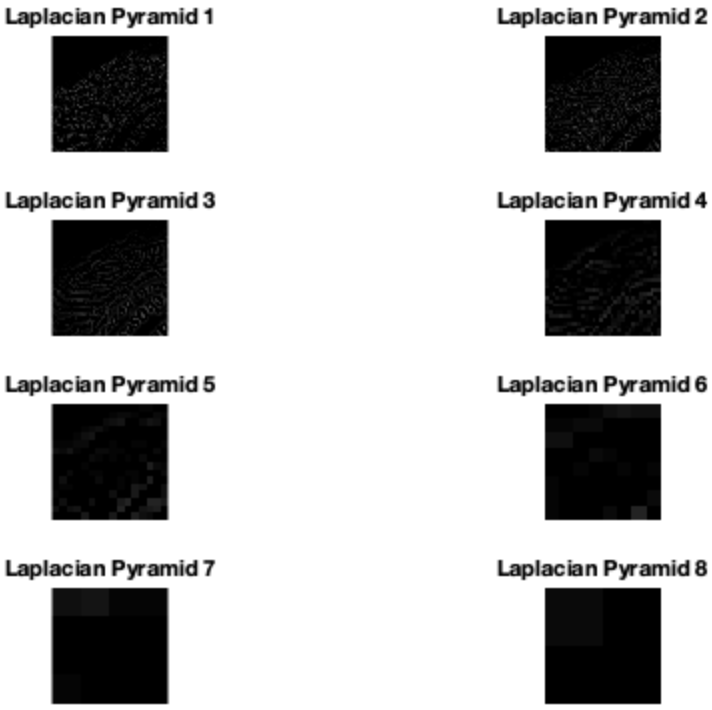
Laplacian Pyramid 4

Laplacian Pyramid 5

Laplacian Pyramid 6

Laplacian Pyramid 7

Laplacian Pyramid 8



Published with MATLAB® R2019b

Part 3: Multi-Scale Edge Detection

```
images = ["../images/CARTOON.jpg", "../images/flowergray.jpg", "../
images/kitty.jpg", ...
    "../images/polarcities.jpg", "../images/text.jpg" ];

filter = [0.25,0.25; 0.25,0.25];
secondDerivative = [-0.125, -0.125, -0.125; -0.125, 1, -0.125; -0.125,
 -0.125, -0.125];

threshold = 5;

for image = images
    % Load image
    img = double(imread(image));
    [height_G, width_G] = size(img);
    side = log2(height_G);

    % Generate the second order derivative images
    temp = imfilter(img, secondDerivative, 'replicate', 'same');
    temp(temp>0) = 1;
    temp(temp<=0) = 0;

    % Plot image
    figure;
    subplot(3, 3, 1);
    imshow(temp);
    title('Original image')

    for i= 1:side
        img = imfilter(img, filter, 'replicate', 'same');
        % Half size
        [height, width] = size(img);
        img = img(1:2:height, 1:2:width);

        secondDer = imfilter(img,
 secondDerivative, 'replicate', 'same');
        secondDer = imresize(secondDer, 2^i, 'bilinear');

        % Detect the zero crossing in the segmented image.
        segment = secondDer;
        segment(secondDer>0) = 1;
        segment(secondDer<=0) = 0;

        % Get all neighbors
        cross = zeros(size(segment));
        for j = 1: height_G
            for k = 1:width_G
                neighbors = getNeighbor(segment, j, k);
                if (~all(neighbors == segment(j, k)))
                    cross(j, k) = 1;
                end
```

```
            end
        end

        localVariance = lclvr(secondDer, cross);

        edgeImage = ones(size(localVariance));
        edgeImage(localVariance > threshold) = 0;

        % Plot nth time result
        subplot(3, 3, 1 + i);
        imshow(edgeImage);
        title(sprintf('%d times filtering', 2^i));
    end
end

function localVariance = lclvr(image, cross)
    [height, width] = size(image);
    localVariance = zeros([height width]);
    for i = 1: height
        for j = 1: width
            if (cross(i, j) == 0)
                continue
            end
            % Compute neighbor's local variance
            neighbors = getNeighbor(image, i, j);
            localVariance(i, j) = var(neighbors(:));
        end
    end
end

function neighbors = getNeighbor(image, i, j)
    index = 0;
    [height, width] = size(image);
    neighbors = ones(1, 9) * image(i, j);
    for x = -1:1
        for y = -1:1
            index = index + 1;
            % Only for inbound elements
            if (inbound(i, j, x, y, width, height) == 0)
               continue
            end
            neighbors(index) = image(i + x, j + y);
        end
    end
end

function isInBound = inbound(i, j, x, y, width, height)
    isInBound = (i + x <= height && j + y <= width && i + x > 0 && j +
 y > 0);
end
```

Original image

2 times filtering

4 times filtering

8 times filtering

16 times filtering

32 times filtering

64 times filtering

128 times filtering

256 times filtering

Original image

2 times filtering

4 times filtering

8 times filtering

16 times filtering

32 times filtering

64 times filtering

128 times filtering

256 times filtering

Original image

2 times filtering

4 times filtering

8 times filtering

16 times filtering

32 times filtering

64 times filtering

128 times filtering

256 times filtering

Original image

2 times filtering

4 times filtering

8 times filtering

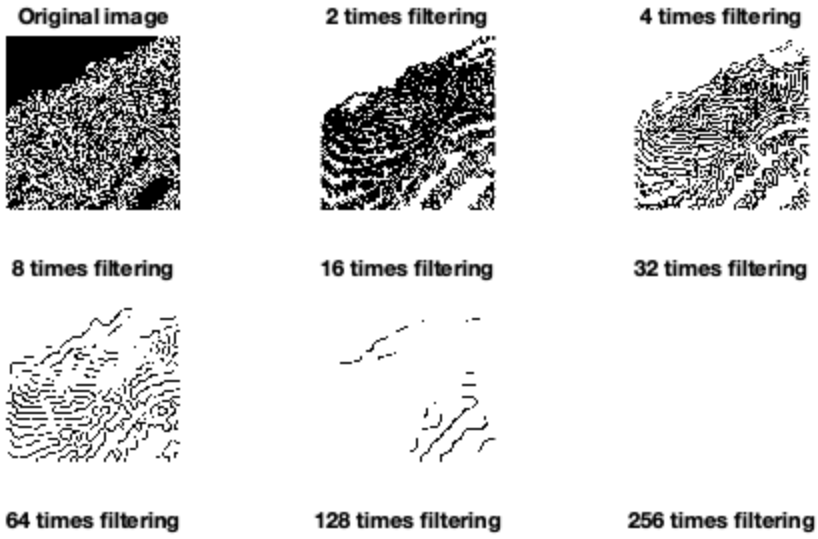
16 times filtering

32 times filtering

64 times filtering

128 times filtering

256 times filtering

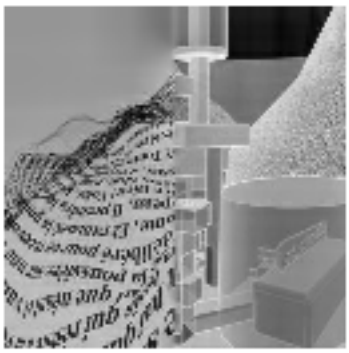


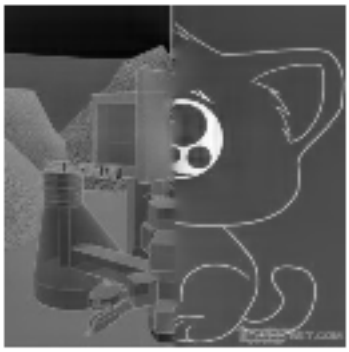
Published with MATLAB® R2019b

Part 4: Multi-Resolution Spline

```
images = ["../images/CARTOON.jpg", "../images/flowergray.jpg", "../
images/kitty.jpg", ...
    "../images/polarcities.jpg", "../images/text.jpg" ];

% define the image pair
n = 3;
image1Index = [4,3,1];
image2Index = [5,4,2];
filter = [0.25,0.25; 0.25,0.25];

for i = 1 : n
    image1 = double(imread(images(image1Index(i))));
    image2 = double(imread(images(image2Index(i))));

    % Blend image
    blendedImage = zeros(size(image2));

    % Get mask
    bi_mask = ones(256,256);
    bi_mask(:,1:128) = 0;

    LP1 = laplacianPyramid(image1, filter);
    LP2 = laplacianPyramid(image2, filter);

    GR = gaussianPyramid(bi_mask, filter);
    LS = {};

    for j = 1: length(LP1)
        LS{end +1} = GR{j}.*LP1{j} + (1-GR{j}).*LP2{j};
        blendedImage = blendedImage + LS{end};
    end

    figure;
    imshow(blendedImage,[]);
end

function lp = laplacianPyramid(img, filter)
    [height, width] = size(img);
    side = log2(height);
    lp = {};

    for i = 1:side
        filteredImg = imfilter(img, filter, 'replicate', 'same');
        [h, w] = size(filteredImg);
        filteredImg = filteredImg(1:2:h, 1:2:w);

        % Get binary image
        biImage = imresize(filteredImg, 2, 'bilinear');

        % Resize
```

```
        lp{end+1} = imresize(biImage - img, [height,
 width], 'bilinear');
        img = filteredImg;
    end
end

function gp = gaussianPyramid(img, filter)
    [height, ~] = size(img);
    side = log2(height);
    gp = {};

    for i = 1:side
        filteredImg = imfilter(img, filter, 'replicate', 'same');
        [height, width] = size(filteredImg);
        filteredImg = filteredImg(1:2:height, 1:2:width);

        % Resize
        gp{end+1} = imresize(filteredImg, 2^i, 'bilinear');
        img = filteredImg;
    end
end
```



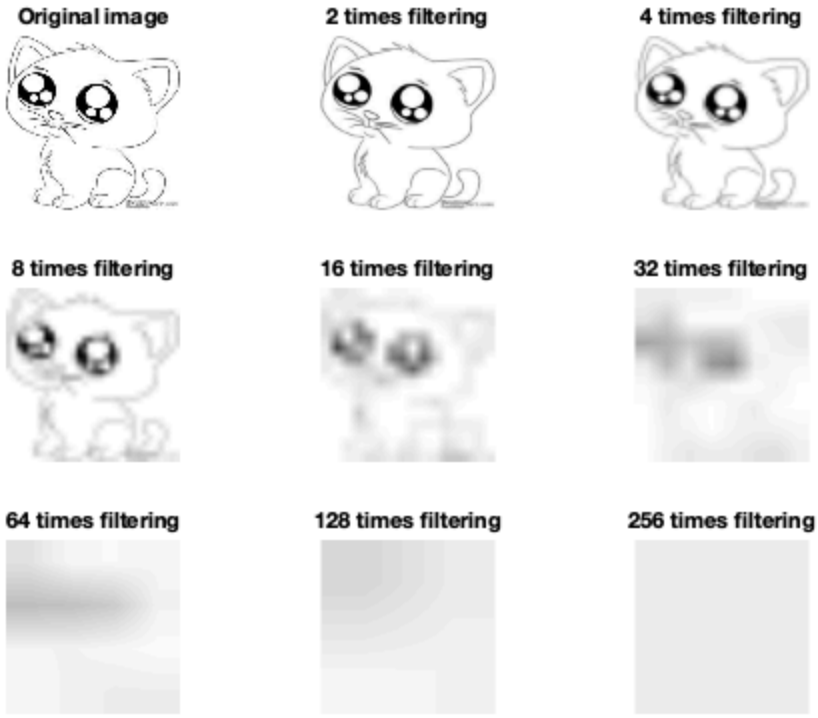


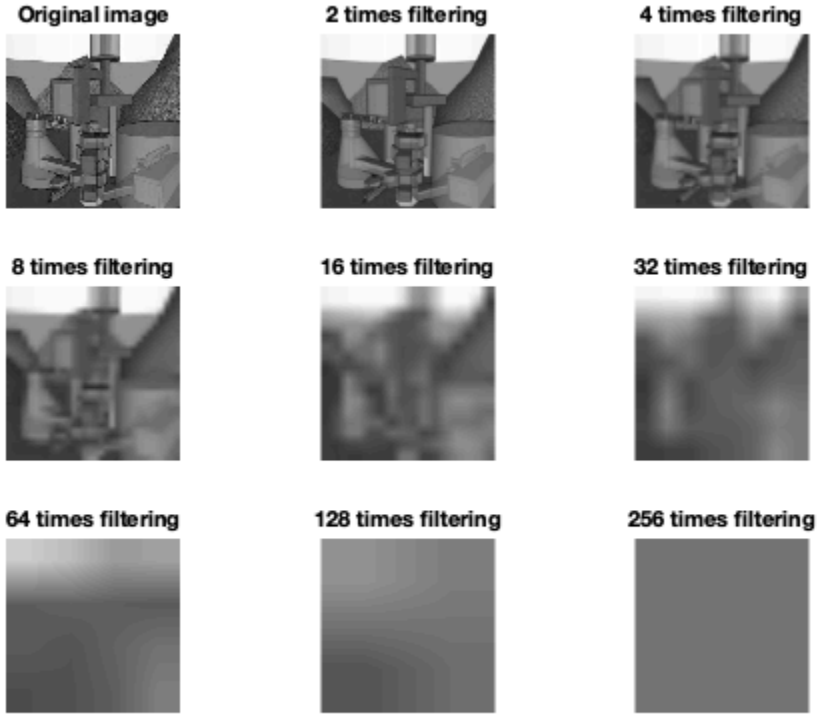
Published with MATLAB® R2019b

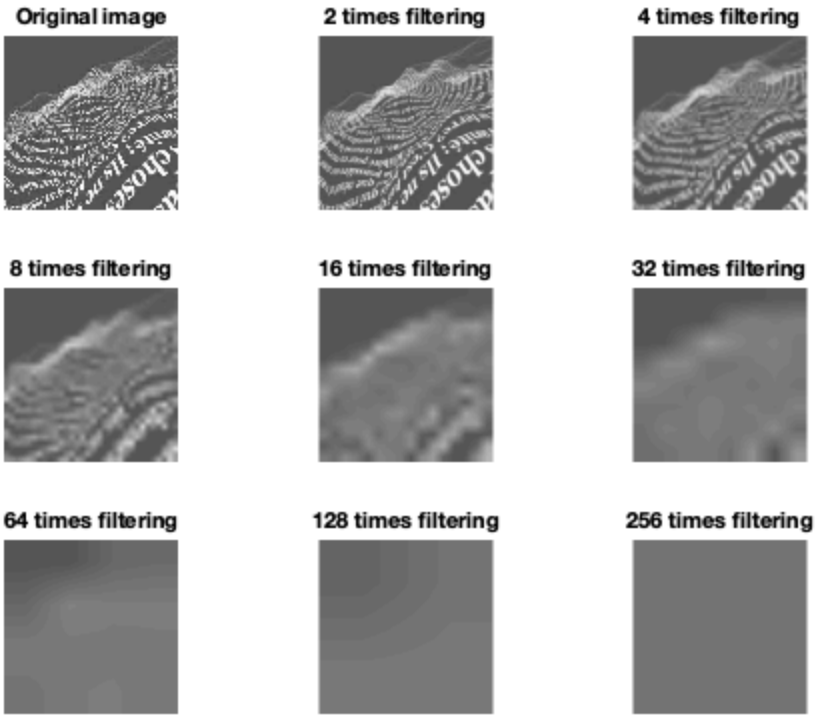
Group

Team Membe: Haixiang Yan & Yang Liu For this assignment, Yang Liu is responsible for part 0, 1 and 2. And Haixiang Yang is responsible for part 3 and 4.

Published with MATLAB® R2019b


Original image
2 times filtering
4 times filtering
8 times filtering
16 times filtering
32 times filtering
64 times filtering
128 times filtering
256 times filtering


Original image
2 times filtering
4 times filtering
8 times filtering
16 times filtering
32 times filtering
64 times filtering
128 times filtering
256 times filtering



Published with MATLAB® R2019b

Group

Team Membe: Haixiang Yan & Yang Liu

For this assignment, Yang Liu is responsible for part 0, 1 and 2. And Haixiang Yang is responsible for part 3 and 4.

Published with MATLAB® R2019b