Biniyam Yemane-Berhane & David Landry  
B EE 341, Lab 2

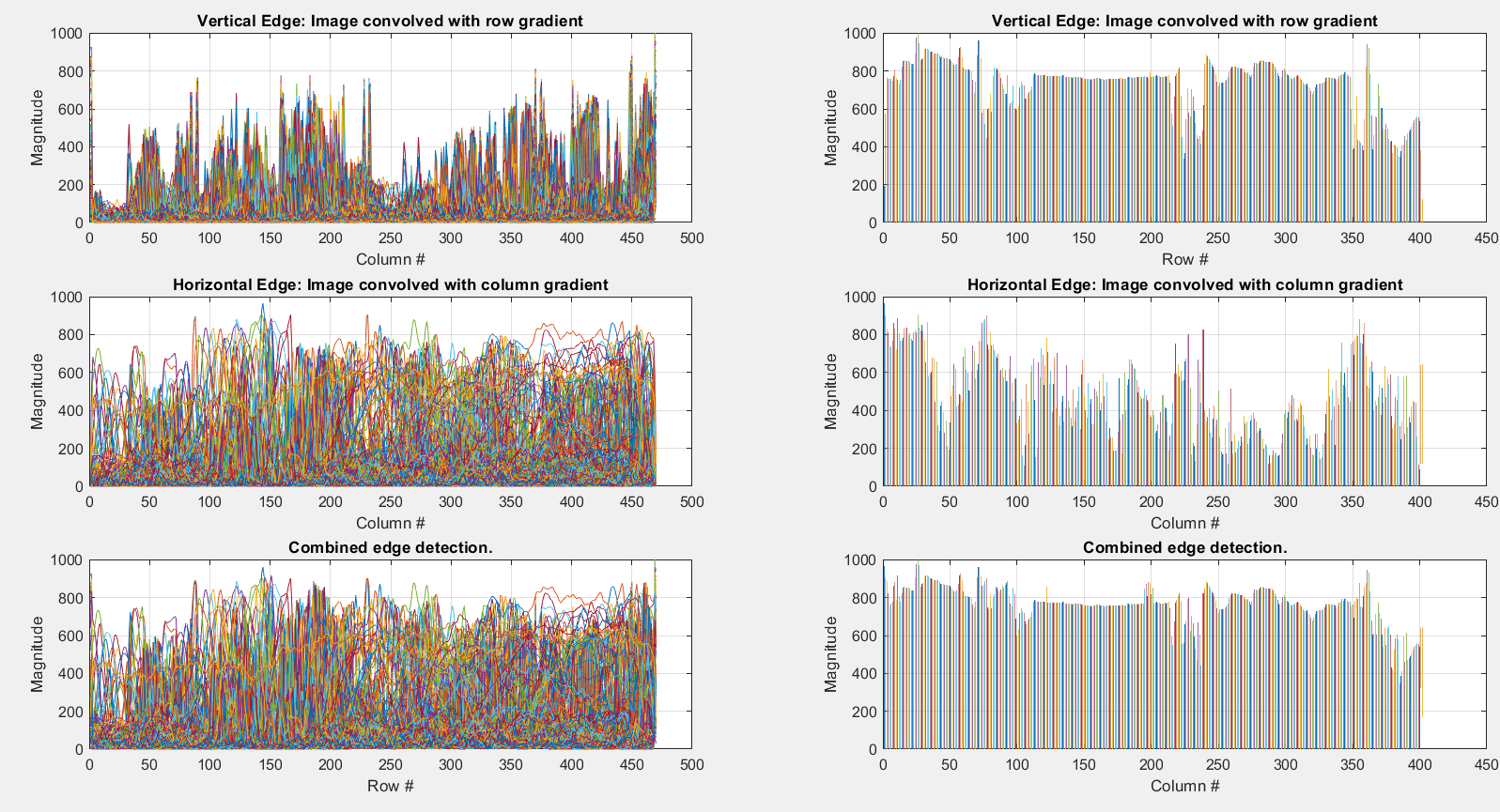
**Lab 2: Introduction to Image Processing**

Part 1: Edge detection

The size of the image is 400 x 468.

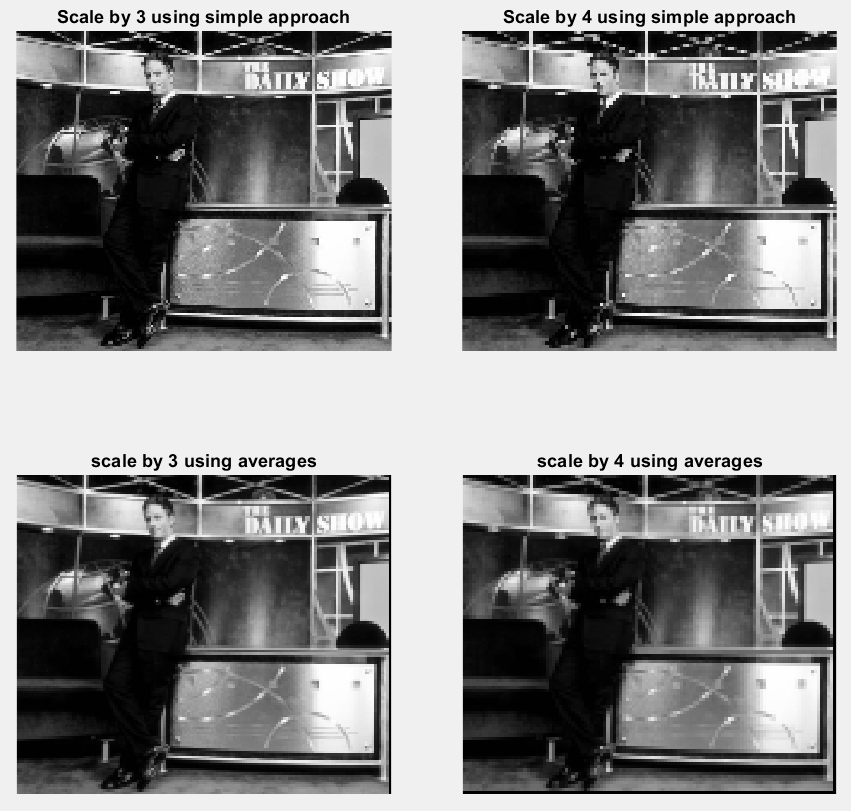


Provided is a plot of the vertical and horizontal edge gradients, as well as the magnitude of the overall gradient.



Part 2: Image Scaling

Here is the combined image of scaling by picking every nth point vs. picking the average of each nth point and its surrounding points:

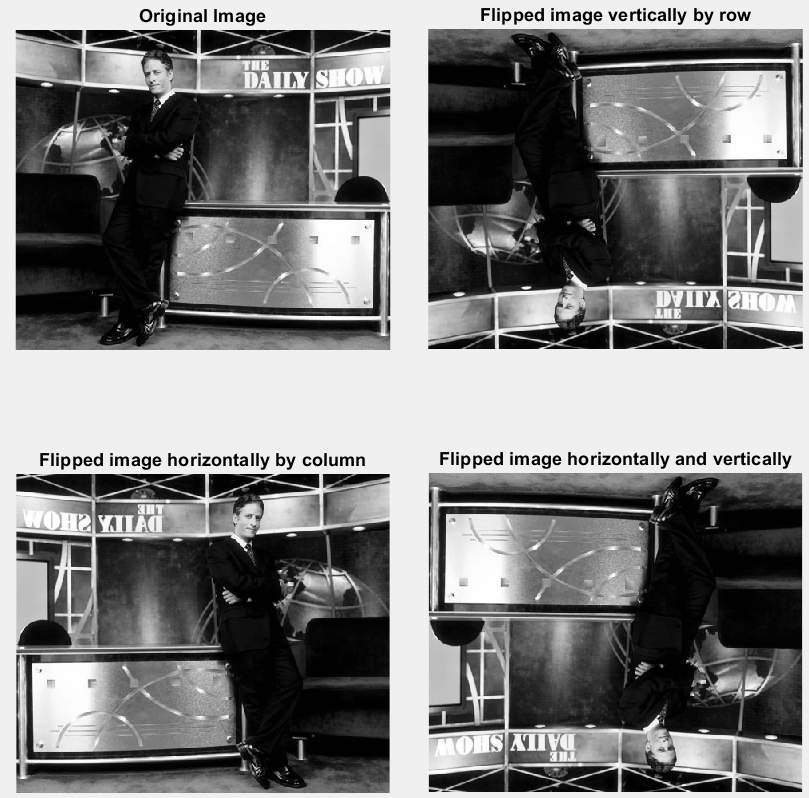


The two average scalings are somewhat smoother than the two scalings by picking each nth point.

The image that was downsized by a factor of 3 had a size of 133 x 156. The image that was downsized by a factor of 4 had a size of 100 x 117.

Part 3: Flipping Images

The first image, x[N-n+1, m] flips the image by rows (vertically).  
The second image, x[n, M-m+1], flips it by columns (horizontally).  
The third, x[N-n+1, M-m+1], flips it both horizontally and vertically.

To verify, we performed flip operations on the original image:  


Part 4: Interpolating Images

For this part, we doubled the image by inserting 1 arbitrary point between each sample point. Interpolation fills in the "in-between" points by taking the average between surrounding sample points. The resulting image has a size of 799 x 935:



Appendix: Matlab code

%--------------------------------------------------------------------------

% Function: Main

% Description: Calls get\_vertical, get\_horizontal, detect\_edge,

% scale\_image\_simple, scale\_image\_averages.

%--------------------------------------------------------------------------

% Obtain the image and convert it to grayscale.

imageMatrix = rgb2gray(imread('DailyShow', 'jpeg'));

% Record the size of the image.

sizeImage = size(imageMatrix);

% Apply the vertical-edge gradient.

verticalEdge = get\_vertical(imageMatrix);

sizeVertical = size(verticalEdge);

magVertical = abs(verticalEdge);

% And apply the horizontal-edge gradient.

horizontalEdge = get\_horizontal(imageMatrix);

sizeHorizontal = size(horizontalEdge);

magHorizontal = abs(horizontalEdge);

% Use the magnitudes to detect the edges of the image.

edgeDetection = detect\_edge(magVertical, magHorizontal);

% Apply 3x and 4x downsampling of the image.

scale3x = scale\_image\_simple(imageMatrix, 3);

scale4x = scale\_image\_simple(imageMatrix, 4);

% Apply downsampling with averages of pixels in sample vicinity.

scale3x\_b = scale\_image\_averages(imageMatrix, 3);

scale4x\_b = scale\_image\_averages(imageMatrix, 4);

% In the 3rd part of this assignment, we are given 3 matrices:

% x[N-n+1, m]

% x[n, M-m+1]

% x[N-n+1, M-m+1]

% These flip the image matrix by rows, by columns, and by both,

% respectively. The row\_flip and col\_flip functions do this to verify our

% assertion.

rowFlip = row\_flip(imageMatrix);

colFlip = col\_flip(imageMatrix);

bothFlip = col\_flip(rowFlip);

% Finally, we will upsample the image, interpolating the extra data points

% using bilinear regression.

scaledImage = upsample(imageMatrix, 3);

% Display figure 1: the original grayscale image and the edge-detected

% image.

figure(1);

subplot(2,1,1);

imshow(imageMatrix);

title('Grayscale of Included Image');

subplot(2,1,2);

imshow(cast(edgeDetection, class(imageMatrix)));

title('Edge-Detected image');

% Display figure 2: the edge detection plots.

figure(2);

plotEdges(magVertical, magHorizontal, edgeDetection);

% Display figure 3: the 3x & 4x downsampling by nth sample and by nth

% average sample.

figure(3);

subplot(2,2,1);

imshow(scale3x);

title('Scale by 3 using simple approach');

subplot(2,2,2);

imshow(scale4x);

title('Scale by 4 using simple approach');

subplot(2,2,3);

imshow(scale3x\_b);

title('scale by 3 using averages');

subplot(2,2,4);

imshow(scale4x\_b);

title('scale by 4 using averages');

% Display the flipped image.

figure(4);

subplot(2,2,1);

imshow(imageMatrix);

title('Original Image');

subplot(2,2,2);

imshow(rowFlip);

title('Flipped image vertically by row');

subplot(2,2,3);

imshow(colFlip);

title('Flipped image horizontally by column');

subplot(2,2,4);

imshow(bothFlip);

title('Flipped image horizontally and vertically');

% Display the upsampled image.

figure(5);

subplot(2,1,1);

imshow(imageMatrix);

title('Original image');

subplot(2,1,2);

imshow(scaledImage);

title('Upsampled Interpolated image');

function verticalEdge = get\_vertical(imageMatrix)

verticalEdgeMask = [-1 0 1; -2 0 2; -1 0 1];

verticalEdge = conv2(imageMatrix, verticalEdgeMask);

function horizontalEdge = get\_horizontal(imageMatrix)

horizontalEdgeMask = [1 2 1; 0 0 0; -1 -2 -1];

horizontalEdge = conv2(imageMatrix, horizontalEdgeMask);

function edgeDetection = detect\_edge(magVert, magHoriz)

edgeDetection = sqrt(magVert.^2 + magHoriz.^2);

%--------------------------------------------------------------------------

% Function: scale\_image\_simple

% Description: Scale the image by selecting each (factor)th pixel,

% starting with pixel # (ceiling of factor / 2).

%--------------------------------------------------------------------------

function scaledImage = scale\_image\_simple(imageMatrix, factor)

startingPoint = ceil(factor / 2);

scaledImage = imageMatrix(startingPoint:factor:end, ...

startingPoint:factor:end);

%--------------------------------------------------------------------------

% Function: scale\_image\_averages

% Description: Scale the image by a factor of (factor) by taking the

% average value of the selected pixel and those in its

% vicinity. It does this by interposing a

% (factor x factor) matrix over the original,

% averaging the pixel values within that vector, and

% copying that value to a set of coordinates of the new

% scaled image.

%--------------------------------------------------------------------------

function scaledImage = scale\_image\_averages(imageMatrix, factor)

% We need a (factor x factor) sub-matrix of the original matrix.

% Initialize the scaled image matrix to a zero matrix to make the program

% run more efficiently.

sizeOfMatrix = size(imageMatrix);

scaledImage = zeros(uint8(sizeOfMatrix(1)/factor), ...

uint8(sizeOfMatrix(2)/factor));

% Counters for row & column number for the new scaled matrix.

scaledRow = 1;

scaledCol = 1;

for i = 1:factor:sizeOfMatrix(1)-factor

for j = 1:factor:sizeOfMatrix(2)-factor

% temp is our interposed (factor x factor) matrix over the original

% image matrix.

temp = zeros(factor, factor);

for k = 0:factor-1

for l = 0:factor-1

temp(k+1,l+1) = imageMatrix(i+k, j+l);

end

end

scaledImage(scaledRow, scaledCol) = mean2(temp);

scaledCol = scaledCol + 1;

end

scaledCol = 1;

scaledRow = scaledRow + 1;

end

% Convert the scaled image to uint8. Otherwise, it will not display

% properly.

scaledImage = uint8(scaledImage);

function rowFlip = row\_flip(imageMatrix)

% Flip it by row (vertically):

rowFlip = flipud(imageMatrix);

function colFlip = col\_flip(imageMatrix)

% Flip it by column (horizontally):

colFlip = fliplr(imageMatrix);

function scaledMatrix = upsample(imageMatrix, scale)

% Get the data type of our original image matrix (i.e., uint8)

dataType = class(imageMatrix);

scaledMatrix = interp2(double(imageMatrix));

scaledMatrix = cast(scaledMatrix, dataType);