Lab 2: Introduction to Image Processing

University of Washington, Electrical Engineering

EE 341

17 July 2018

|  |  |  |
| --- | --- | --- |
| Kalvin Hallesy 1750416 | Irina Golub 1775424 | Chenglong Li 1429842 |

# 

# 1. Introduction

The purpose of this lab is to become familiar with a few basic image processing concepts. Sections 2 and 3 describes the process of performing a simple edge detection on an image. Section 4 describes two methods for scaling down an image to create a thumbnail - the first by selecting a subset of the image pixels, and the second by averaging the values in a handful of pixels. In section 5, we discuss how the rows and columns of a matrix can be reversed to flip an image vertically, horizontally, or both ways. Finally, in section 6 we turn an NxM image into a 2Nx2M image using linear interpolation.

# 2. Edge Detection

Edge detection is often a first step used in more complicated image processing operations. The result of an edge detection shows changes or discontinuities in an image amplitude attribute such as luminance. We used the Sobel row gradient operator and column gradient operator to detect horizontal and vertical edges in an image.

The edge gradient image contain many dark areas. To possibly save toner if we are required to print these images, we reversed the grayscale of each image such that the darkest area is transformed to the brightest area and vise versa.

## 2.1 Implementation

First we read the image using the ‘imshow’ function and converted it into a grayscale images using the ‘rgb2gray’ command. The original image as well as the resulting grayscale image is shown in Fig. 1, with a size of 400 x 468 pixels.



  
Fig. 1: Original Image (upper) and Grayscale Image(lower)

The image was convolved with the Sobel vertical and horizontal edge detection convolution kernels using the conv2() function. Both matrices in each conv2() instantiation were converted to type ‘double’ for the purpose of using the convolution function properly.

The result of the two convolutions was scaled down to hold only values between 0 and 255 by using ‘max(max(ResultingMatrix))’ and multiplying the matrix by 255 / maximum value. This ensures that the new matrix values will be properly decoded as colors of the edge detection image.

The row gradient image was displayed by taking the absolute value of the horizontal edge detection matrix, converting it to ‘uint8’, and displaying the image using ‘imshow’. A similar process was completed to display the column gradient image, but instead using the vertical edge detection matrix. The overall gradient was found by taking the square root of both matrices squared and added together.

To invert the grayscale of the image, we wrote a ‘reverse’ function that takes the image as an input. A new image matrix is defined with the same size as the input image. We loop through each element of the new matrix using a for loop from 1 to the last pixel (rows\*columns of input). Inside the for loop, the ith pixel is defined as 255 subtracted by the value of the ith pixel of the original image. This way, black(0) becomes 255 - 0 = 255 (white), and white(255) becomes 255-255 = 0 (black).

## 2.2 Results

The resulting horizontal edge gradient is shown in Fig. 2. The resulting vertical edge gradient is shown in Fig. 3. The overall gradient is shown in Fig. 4. These images match the expected results of the Sobel edge detection method on the original image.

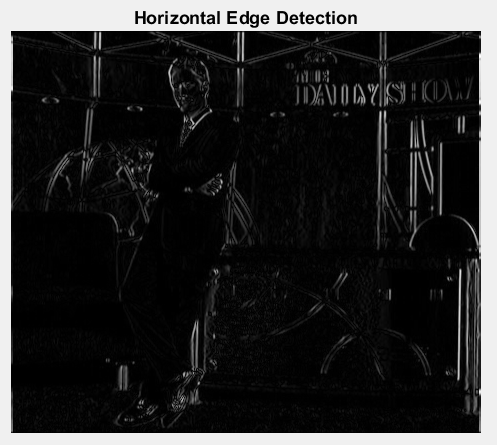
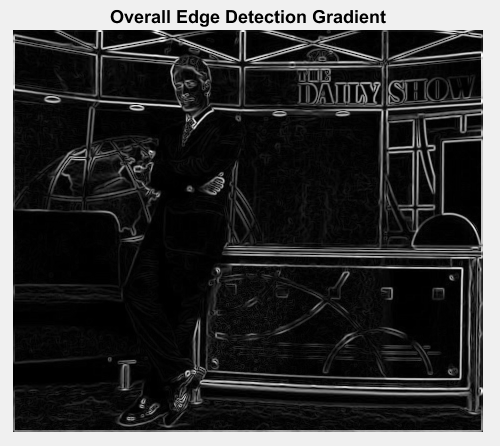
  
Fig. 2: Horizontal Edge Gradient



Fig. 3: Vertical Edge Gradient

Fig. 4: Overall Edge Gradient

The overall edge gradient image was input to the ‘reverse’ function to transform the black color to white and white color to black. The result is shown in Fig. 5.



Fig. 5: Overall Edge Gradient With Reversed Colors

Based on visual inspection, this is the expected result - all black color turned white, white color turned black, and colors in between these two shifted to the opposite shade.

# 3. Edge Detection on Another Image

To better understand the horizontal and vertical gradients, we decided to complete another edge detection on an image of fire, shown in Fig. 6.



Fig. 6: New Image of Fire

This image has a black background with bright orange flame, so we predicted that it would contain a lot of horizontal and vertical edges. We applied the horizontal , vertical, and overall edge detector on this image and graphed the edge gradient magnitudes as in Part 2.

## 3.1 Implementation

The Sobel edge detection method was applied to this image in the same way as in Part 2 and using the same code. After converting the image to grayscale, the image was convolved with the horizontal and vertical edge detection kernels. The resulting matrices were scaled to hold values from 0 to 255. The absolute value of both matrices was displayed as well as the square root of the sum of both matrices squared.

## 3.2 Results

The original grayscale image is shown in Fig. 7. Fig. 8 shows the resulting horizontal edge gradient, Fig. 9 shows the resulting vertical edge gradient, and Fig. 10 shows the overall edge gradient.



Fig. 7: Grayscale Version of Chosen Image



Fig. 8: Horizontal Edge Gradient

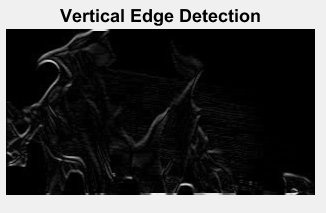


Fig. 9: Vertical Edge Gradient

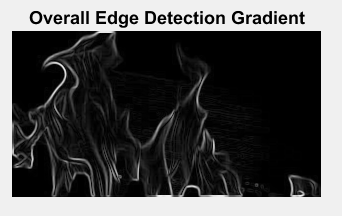


Fig. 10: Overall Edge Gradient

The results in these images follow the trend from Part 3 of edge detection on another image. These images are an accurate portrayal of changes in the image amplitude.

# 4. Scaling Down an Image

The next area of image processing we explored was changing physical properties, such as size and orientation, of the overall image. We started with downsizing the image. This involves taking the image and a desired scaling of the original size and outputting a lower resolution version of the same image such that it contains a smaller area of pixels.

We explored two seperate techniques to accomplish this downscaling, pixel selection and pixel averaging. Pixel selection involves developing an algorithm to select evenly spaced pixels throughout the image while discarding the unselected pixels to be left with the final image. Pixel averaging involved choosing multiple areas of pixels and averaging their brightness values together to create the pixels for the final image.

## 4.1 Implementation

For the pixel selection method, we created a function called scaleDown (see appendix). This function creates the smaller image by creating a new matrix with pixels each spaced by a scale factor on the original image. Therefore, for even scale factors, each pixel spacing would contain an even amount of pixels so we decided to just take the first pixel of every block. For odd scale factors, each pixel block would have an odd spacing which allowed for the center pixel to be selected instead.

For the pixel averaging method, we created a function called scaleDownAverage. Our averaging algorithm requires the same amount of pixels in every block so we started by padding the image with additional pixels if the dimensions weren’t multiples of the scaling factor. We then used a quadruple nested for loop to parse the image into individual blocks that had the length and width of an sFactor. Within each of these blocks, each individual pixel is parsed and added to a running total called sum. After the block is completed, the sum is divided by the amount of pixels analyzed to get the average and that value becomes the value for the pixel of that location on the downscaled image. Sum is then reset to zero and the process continues on the next block.

## 4.2 Results

Both methods successfully produced downscaled versions of the original image whilst retaining a reasonable amount of the original image quality. However, each method had its own pros and cons.

### 4.2.1 Pixel Selection Method

The pixel selection method ended up being extremely good at preserving the brightest and darkest colors in the image. The resultant image also is extremely sharp. This can be seen in figure 11 and 12 which show the resultant downscaling by 2 and 5 respectively of the Daily Show image.



Fig. 11: Downscaled by 2 via Pixel Selection

Upon an initial glance, pixel selection downscaling by 2 yields a very compelling recreation of the original in color and sharpness. The only complaint is some aliasing on the edges of the face and other diagonal lines which gives a sort of ‘cardboard cut-out’ appearance.



Fig. 12: Downscaled by 5 via Pixel Selection

A downscaling of 5 leads to a much more drastic loss in quality, the aliasing along the diagonal edges and face now turns into a blocky mess and much of the detail is lost. For example, the sign is now completely illegible.

### 4.2.2 Pixel Averaging Method

Pixel averaging offers very different qualities in comparison to pixel selection. The color resolution becomes a lot less varied, the whites are darker and the blacks are lighter. The sharpness is also much softer and leads to the edges sort of blurring with their surroundings; however, this allows for much less aliasing and more detail to be preserved. These qualities are shown in figures 13 and 14 which show the Daily Show image downscaled by 2 and 5 respectively via pixel selection.



Fig. 13: Downscaled by 2 via Pixel Averaging

An initial glance, this looks very similar to the image produced by the pixel selection method; however, the details are more apparent in a close look. This image doesn’t suffer from the harsh sharp-edge aliasing of the pixel selection method which results in a more natural looking face and diagonal lines. The color washing effect is also not present due to the small areas of pixels needing to be averaged at a scale of 2.



Fig. 14: Downscaled by 5 via Pixel Averaging

As opposed to the scaling of 2, the scaling of 5 in averaging produces a drastically different image when compared to selection. The color is much more washed out and harsh differences in color seem to blend together into grey. This leads to the edges also being much softer. However, the detail is much more preserved: the sign of the Daily Show is still readable and the facial expression is more closely matched to the original.

### 4.2.3 Choosing a Superior Method

Because each method excels in different areas, consideration of the situation and desired result is needed when choosing which method to use.

For low to moderate scaling such as a scaling factor of two, averaging is almost always better. The negative effects from averaging such as color washing and edge blurring are almost nonexistent at lower scaling factors due to the individual pixel blocks being much smaller so there is no downside. On the other hand, selection produces lines that are overly sharp and therefore create unnecessary aliasing.

For higher scaling factor values such as five, there is no clear winning method and are image dependent. If high color resolution and sharp edges are desired, selection will produce a much sharper and more vibrant image. However, if details are more important there is less lost in averaging despite the washed out colors and blurriness.

# 5. Flipping an Image

Since an image is represented with a 2D matrix, flipping an image requires re-arranging the matrix in certain ways. First, we are asked to predict how the following images look when compared to the original image X[n,m]:

I. X[N-n+1,m]: We predicted that this would be the vertically-flipped version of the image. This is because only the rows are being altered in this matrix.

II. X[n,M-m+1]: We predicted that this would be the horizontally-flipped version of the image because only the columns, m, are being altered in this version.

III. X[N-n+1,M-m+1]: We predicted this would be both a vertical and horizontal flipped version of the image because rows and columns are both altered.

## 5.1 Implementation

To test our predictions, we used matrix manipulation to create image I, II, and III manually. This was done using two nested for loops - one traversing rows, and another traversing columns. Inside the nested loops are the three statements for making the three new images: vFlip(n,m) = grayPic(N-n+1,m) and so on.

We then plotted the flipped images using the ‘fliplr’ function for horizontal flipping, ‘flipud’ function for vertical flipping, and both for a horizontally and vertically flipped image. The results were compared to the original images to verify if our predictions were accurate.

## 5.2 Results

The results of the first method of flipping, matrix manipulation with for loops, is shown in Fig. 15 in the order of horizontally, vertically, then both.

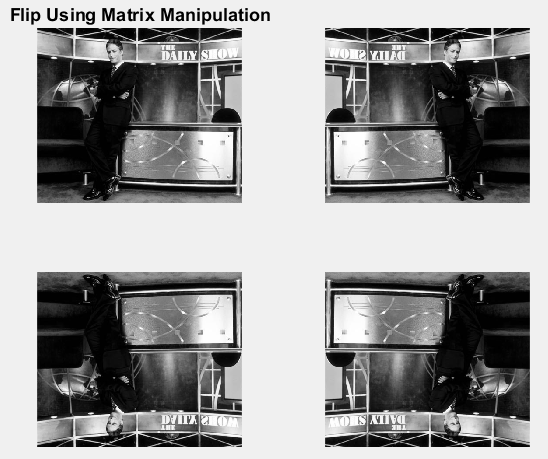


Fig. 15: Results of flipping the image using matrix manipulation.

Based on this result, our predictions were correct: X[N-n+1,m] is a horizontal flip, X[n,M-m+1] is a vertical flip, and X[N-n+1,M-m+1] is both.

The flipped images using MATLAB functions were displayed in corresponding locations as shown in Fig. 16.



Fig. 16: Results of flipping the image using predefined functions, placed in locations corresponding to our predictions.

Our predictions are further verified to be accurate in that the output of the predefined horizontal flipping function matched our prediction of horizontal flipping matrix manipulation, and the same was true of the vertical flipping and both-direction flipping functions.

# 6. Scaling Up an Image

To scale an image to a larger size, new pixels in between the original pixels have to be computed. We computed these pixels using linear interpolation - the new samples become the midpoint value of the previous samples. This method was used to turn an image with dimensions NxM into an image with dimensions 2Nx2M.

## 6.1 Implementation

To do the scaling, we wrote a function that takes a grayscale image as the input. The image is converted to type double and input to MATLAB’s ‘interp2’ function with specifying keyword ‘linear’. The output of interp2 is converted back to type uint8 and becomes the output of the function - the image with dimensions twice those of the original image.

## 6.2 Results

We called the function with a grayscale image: the original image and the function output are shown in Fig. 17.

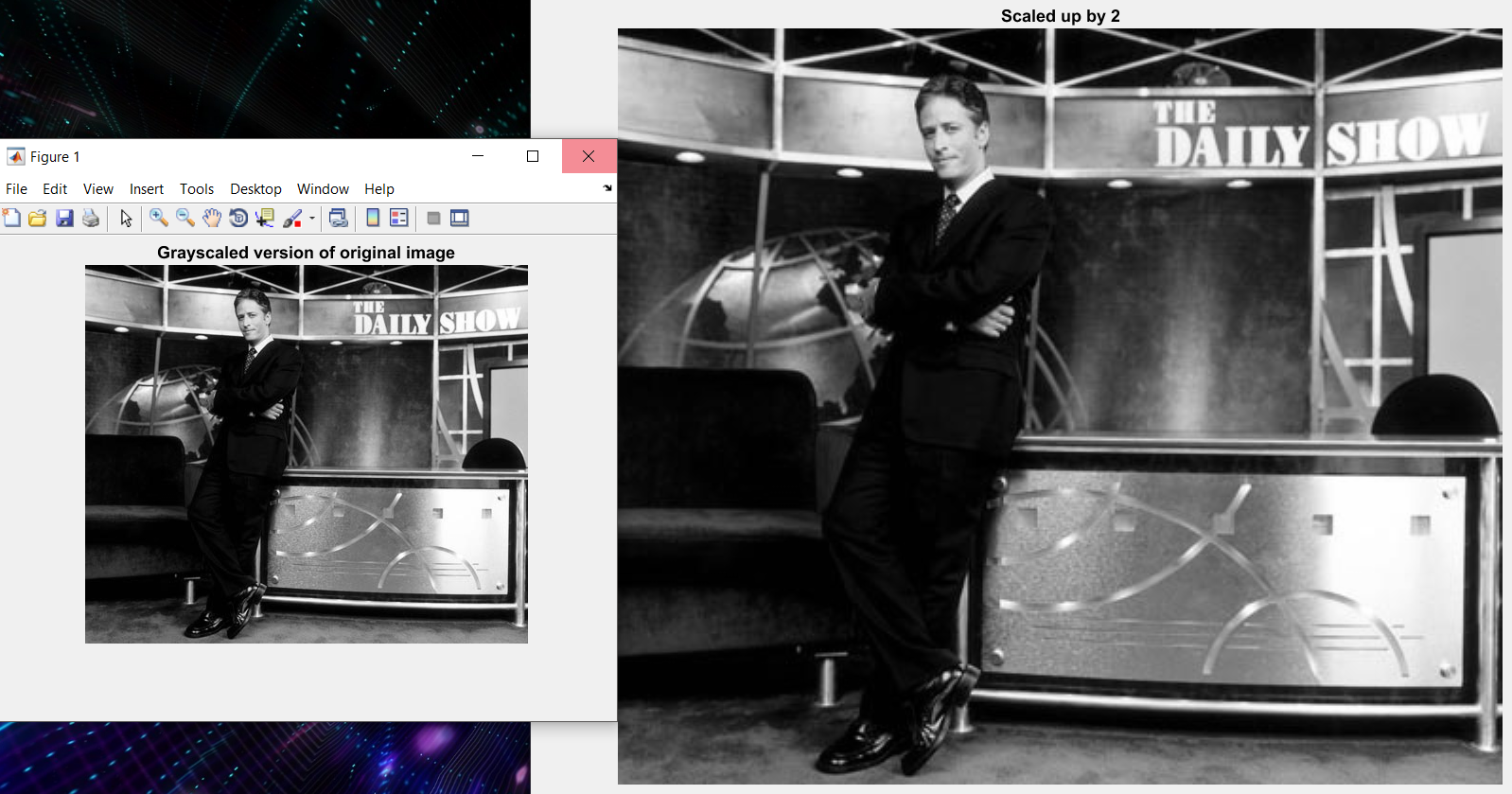


Fig. 17: Original image (left) and scaling function output (right)

The image output of the function looks to be the same as the original image, but twice the size. We can verify this by displaying the size of the input and output image:

ans =

400 468

ans =

799 935

These outputs show that the height of the output is approximately twice that of the input and the width of the output is twice the input as well. These results match the desired results.

# 7. Conclusion

MATLAB is a valuable tool that can be used for image processing applications. It can be used to perform the edge detection on an image by convolving with the Sobel edge detection kernels, to shrink down an image using various different methods, to flip the direction of an image using matrix manipulation or predefined MATLAB functions, and to expand an image using linear interpolation.

## 5.1 Difficulties

Overall, we faced a good amount of difficulty in understanding the lab instruction and and specifics of image processing. In Assignment 1, we had trouble properly displaying the edge detection gradients until we were told to scale the edge detection results from 0 to 255 and convert them to type uint8.

In Assignment 3, we had a lot of trouble writing a function that would take the average value of each pixel in a block with dimensions of the scaling factor. Once we figured that out, we struggled to display the resulting image properly before realizing that the variable storing the sum was too small and should be converted to type uint16.

## 5.2 Final Thoughts and Discussion

In this lab we learned a lot about image processing applications. We were also able to focus on the importance of variable type in using functions and displaying images. One way we would like to build on this knowledge is to study how filters are applied to images, mathematically - for example, a smoothing filter. Also, we would like to explore other methods of interpolating pixels to improve the quality of an image. These two are just the beginning of an enormous variety of image processing applications and techniques that are very relevant to the modern world.

# Appendix A

In a zip folder submitted with this report, please find the .m-file for the following sections and functions:

* Part 1 edge detection: “Assignment1.m”
* Part 1 color transformation function: “reverse.m”
* Part 2 edge detection: “Assignment2.m”
* Part 3 scaling function by selection: “scaleDown.m”
* Part 3 scaling function by averaging: “scaleDowAverage.m”
* Part 3 function testing: “Assignment3.m”
* Part 4 flipping: “Assignment4.m”
* Part 5 scaling function “scaleUpBy2.m”
* Part 5 function testing “Assignment5.m”