Group 25 - Alex Breda, Antony Devita, and David Fields

Mawutor Kofi Amanfu

MATH 1554

21 July 2020

Final Project Report

1. Introduction

The problem our project addresses is figuring out how to blur the faces in digital images using techniques from Linear Algebra. The solution to this problem can be found by applying one of a few different filters to the images. The models that are involved in our project are the matrices that are used in the filters. The solutions to our problem are the results that we receive after we apply the filters to the images.

We addressed our problem and worked towards solving it by writing two programs in Python. The two programs utilize the OpenCV library for Python. Our first program, kernel.py, is useful for demonstrating the models that are involved in our problem. Our second program, blur-image.py, is used to find the solutions to our problem.

2. Problem of Interest and the Model

The first program, kernel.py, is used to illustrate how we can approach the problem and the models that are involved. First, the program prompts the user to input the file name of the image with the faces and stores the input. Next, the program prompts the user to input which blur filter the program should use and stores the input. The three blur type options are Blur, GaussianBlur, and MedianBlur. Next, in order to blur the faces in the digital image, we need a way to identify them and store their coordinates. The program does this using a method from the

OpenCV library. After storing the coordinates of the faces, the program is ready to apply the blur filter that the user selected. This is the main part of the program which involves Linear Algebra. The code for each of the filters is under an if statement, so the program only runs the code for the filter that the user chose.

The first blur type in kernel.py is Blur. The code for Blur consists of a for loop that runs once for each face that was detected. Inside the for loop, there are two more nested loops. For a particular three by three matrix, the program first stores the values of the nine elements of the matrix. The program then finds the average value of the pixels within the matrix. The program then sets the value of the element in the center of the matrix equal to the average value. This uses kernels, which "are small matrixes used to alter an image in some way by convoluting between the image and the kernel" (Pulfer 5). The program also prints out the average value for demonstration purposes. It performs these calculations for each pixel in the photo. A mathematical representation of this is as follows:

$K = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$ represents the box filter kernel for Blur.

The second blur type is GaussianBlur. The GaussianBlur method is notably more complex than the standard Blur due to its usage of a Gaussian kernel instead of a normal box kernel. The Gaussian kernel uses standard deviation to compute the kernel which in effect creates a more smoothed image instead of a blurry one since it controls for a noticeable color change that would occur at an edge. Due to the complexity of the calculation of standard deviation based on past computation demonstrating this would require extensive knowledge of algorithms, and even then, it is difficult to see how the kernel is able to scan and build a filtered image. The gif below shows a visual representation of how kernels scan and build an image.

Source: https://datacarpentry.org/image-processing/06-blurring/

The third blur type in kernel.py is MedianBlur. This consists of a for loop that runs once for each face that was detected. Inside the for loop, there are two more nested loops. For a particular three by three matrix, the program first stores the values of the nine elements of the matrix. The program then finds the median value of the nine elements. The program then sets the value of the element in the center of the matrix equal to the median value. The program also prints out the median values for demonstration purposes. The mathematical representation of the kernel matrix is very similar to that of Blur, but it calculates the median value through algorithmic logic built into the method definition.

As stated previously, the kernel.py program is useful for illustrating the model for our problem. The main limitation of this program is that it only uses three by three matrices instead of much larger matrices. As a result, the kernel.py program is not very effective at blurring the faces in the image. Effectively, this program is a smaller scale implementation of our second program, blur-image.py.

3. Solutions to the Model

We found the solution to our problem using our second program, blur-image.py. The blur-image.py program is similar to the kernel.py program. However, the blur-image.py program blurs the faces in the image much more effectively. It achieves this because it uses larger matrices. The blur-image.py program is able to use larger matrices in part because it implements the different filter types using methods from the OpenCV library, which the kernel.py program generally did not.

First, the blur-image.py program prompts the user to input the file name of the image with the faces and then stores the input. Next, the program prompts the user to input which blur type the program should use and then stores the input. The four blur type options are Blur, GBlur, MedianBlur, and BFilter. Next, the program uses a method from the OpenCV library to detect the faces in the image and then store their coordinates. Next, there is a for loop in which the program uses a method from the OpenCV library to construct a rectangle encompassing each face. Next, the program implements the blur type that the user previously selected in order to blur the faces. As with the kernel.py program, this is the main part of the program which involves Linear Algebra. The code for each of the blur types is under an if statement, so the program only runs the code for the blur type that the user chose.

The first blur type in blur-image.py is Blur. The program first prompts the user to input an integer n, which corresponds to the dimensions nxn of the kernel matrix that will be used. Next, there is a for loop that runs for each face. Inside the for loop, the program uses the blur method from the OpenCV library to blur the current face. This blur type works "by convolving the image with a normalized box filter" (Mordvintsev and Abid K.). In order to do this, the program “simply takes the average of all the pixels under the kernel area and replaces the central element with this average” (Mordvintsev and Abid K.).

The second blur type in blur-image.py is GBlur. The program first prompts the user to input an integer n, which corresponds to the dimensions nxn of the kernel matrix that will be used. Next, there is a for loop that runs once for each face. Inside the for loop, the program uses the GaussianBlur method from the OpenCV library to blur the current face. GBlur uses a Gaussian kernel, which is different from Blur which uses “a box filter consisting of equal filter coefficients” (Mordvintsev and Abid K.).

The third blur type in blur-image.py is MedianBlur. The program first prompts the user to input an odd integer greater than one, which increases the strength of the blur effect the larger the value is. Next, there is a for loop that runs once for each face. Inside the for loop, the program uses the medianBlur method from the OpenCV library to blur the current face. This method “computes the median of all the pixels under the kernel window and the central pixel is replaced with this median value” (Mordvintsev and Abid K.).

The fourth blur type in blur-image.py is BFilter. The program first prompts the user to input an integer, which increases the strength of the blur effect the larger the value is. Next, there is a for loop that runs once for each face. Inside the for loop, the program uses the bilateralFilter method from the OpenCV library to blur the current face. The following is an explanation of bilateral filtering:

> The bilateral filter also uses a Gaussian filter in the space domain, but it also uses one more (multiplicative) Gaussian filter component which is a function of pixel intensity differences. The Gaussian function of space makes sure that only pixels are ‘spatial

neighbors' are considered for filtering, while the Gaussian component applied in the intensity domain (a Gaussian function of intensity differences) ensures that only those pixels with intensities similar to that of the central pixel ('intensity neighbors') are included to compute the blurred intensity value. As a result, this method preserves edges, since for pixels lying near edges, neighboring pixels placed on the other side of the edge, and therefore exhibiting large intensity variations when compared to the central pixel, will not be included for blurring. (Mordvintsev and Abid K.)

4. Interpretation and Analysis of the Solution

While the kernel.py program does not blur the image effectively, the blur-image.py program does. Each of the four different blur types in blur-imag.py have their own advantages and disadvantages. The advantage of using GBlur is that "Gaussian filtering is highly effective in removing Gaussian noise from the image" (Mordvintsev and Abid K.). The advantage of using MedianBlur is that it "is highly effective in removing salt-and-pepper noise" (Mordvintsev and Abid K.). The advantage of using BFilter is that it "is highly effective at noise removal while preserving edges" (Mordvintsev and Abid K.). A downside of using BFilter, however, is that it "is slower compared to other filters" (Mordvintsev and Abid K.).

5. Conclusion

This project illustrates how Linear Algebra can be used in the field of Computer Science to solve practical problems. We not only explored how Linear Algebra can be used to blur the faces in digital images, we also wrote two programs that modeled and implemented a solution to this problem. Blurring faces is just one example of how Linear Algebra can be used in the field of Computer Science. There are many other applications of Linear Algebra in the field which can also be implemented to solve problems using programming.

Works Cited

"Blurring Images." *Blurring Images – Image Processing with Python*, Data Carpentry, 2016, datacarpentry.org/image-processing/06-blurring/.

"Image Filtering." *OpenCV*, docs.opencv.org/3.4.8/d4/d86/group__imgproc__filter.html.

Mordvintsev, Alexander and Abid K. "Smoothing Images." *OpenCV-Python Tutorials*, 2013, opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_filtering/py_filtering.html.

Pulfer, Erich-Matthew. *Different Approaches to Blurring Digital Images and Their Effect on Facial Detection*. BS Thesis. University of Arkansas, 2019. *ScholarWorks@UARK*, scholarworks.uark.edu/cgi/viewcontent.cgi?article=1067&context=csceuht.