HW3 – IMAGE TRANSFORMATIONS FOR DATA AUGMENTATION

This assignment was used to teach simple image transformations, using some built-in functions to implement certain transformations while having to implement our own algorithms for other transformations. The program is able to transform the same image in eight different ways by calling several helper functions created for this assignment.

10/12/2021

Ryan Orf
CS4650: Digital Image Processing

For this assignment, students were challenged to implement eight different image transformations, using built-in functions for some of the transformations while requiring implementation from scratch for others. Since I could not get Python to work with CV, I have again implemented this assignment in C++, with a make command "./transform". In order to accomplish the desired transformations, I developed a helper function to carry out each transformation. Translation, crop and scale/resize, vertical-flip, horizontal-flip, and rotate were able to use built-in functions. However, random erase, random intensity stretching, and blurring required me to implement my own interpretation of certain algorithms. By running the program five times, the user is able to see five completely different effects of each transformation, save for flipping over an axis.

Due to a lack of statistical results from experimentation, this section will discuss more in-depth the step-by-step process of the code. The main program first ensures only two parameters are given: the command and the input image. Then, the image is read into many different variables pertaining to the different types of transformations so as to be warped with each function call. For translation, a helper function is called to calculate two random values between zero and the number of rows/columns that will determine how much the image is translated. Then, the function creates a matrix to assist in translation, calls a built-in function to translate the image, and returns the translated image to the main function. Similarly, another helper function generates four random, calls a built-in function to crop the image, another built-in function to resize the image, and returns the cropped and rescaled image to the main function.

Vertical and horizontal flipping are extremely straightforward. In fact, these two transformations were implemented using the same helper function, with a flag for the axis as a parameter to prevent redundant reuse of code. Once called from main, this helper function to flip

the image simply calls another built-in helper function to do so and returns the result to main. The helper function to rotate the image, when called, calculates the center of the image and a random angle before referencing a built-in function to calculate the rotation matrix. Finally, the image is rotated using a separate external function, and the final product is returned to main. The random erase function undergoes a process extremely similar to the cropping function, except the values in the range that is randomly generated are turned black. All of this is encompassed by an outer loop to loop through the process correlating to the number of areas you want erased. In fact, while this value is passed by value into the function, the value is declared as a symbolic constant before main in case there was a reason to change the desired number of areas to erase. Of course, the image with the erased sections is returned to main.

The random intensity stretching function calculates a random integer. If odd, the input image is converted to grayscale, or else it remains a colored image. If the image remains in color, the following process will have to be looped three times corresponding to the number of channels. The rest is comparable to the first assignment: random maximum and minimum values are determined, the image is looped through pixel by pixel, a temporary value is calculated for brightness and contrast (with edge cases being considered to prevent overflow), and the pixel value is set equal to the calculated temporary value. The image is then returned to main.

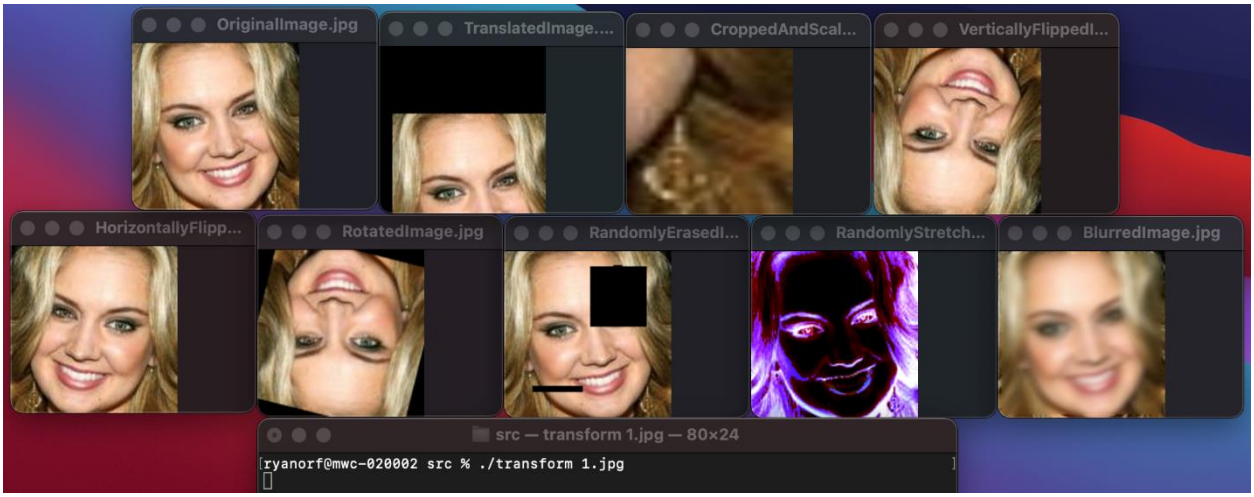
For the final transformation, a box filter had to be applied to simulate a blur. The function randomly calculates whether the normalized matrix of ones will be 3x3, 5x5, or 7x7. Another interesting thing the function does here is create a change variable that is set equal to one if the matrix is 3x3, two if 5x5, or three if 7x7. The matrix is then declared and initialized to values of one divided by the size. Now, each pixel can be looped through, and for each pixel, we have to check all eight surrounding pixels in order to calculate the local average for the blur. The change

variable is used here, as the number of pixels we have to visit varies depending on the size of the blur matrix. Averages for each pixel are then calculated, of course for all three channels, and the image is returned to the main function. Now, all of the results, along with the original image to compare, are displayed to the user.

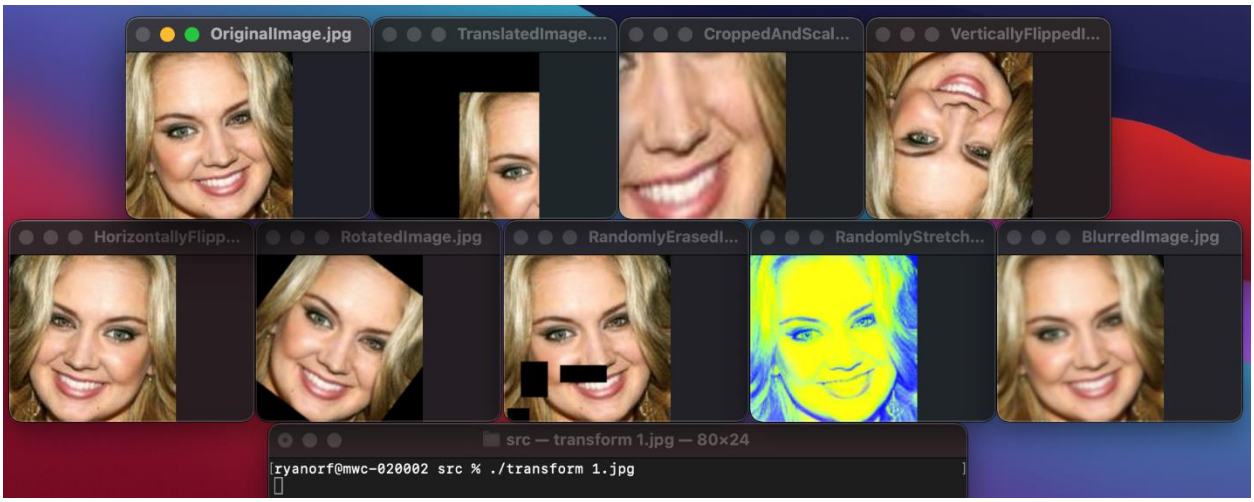
This assignment was much easier to implement than the previous assignment, simply due to the ability to use built-in OpenCV functions. However, the algorithm for implementing the box filter took a while for me to figure out, along with minor changes I made throughout to upgrade the performance or prevent typing the same code. I would like to note that this program is my own creation, however I did learn about and pull ideas from sources I found online when it came to implementing the built-in OpenCV functions I have never used before. In order to prevent academic dishonesty, I have included these resources in my code as comments corresponding to the applicable function. One potential misstep that may have been taken during this assignment involves how many transformations should be calculated each time the program is run. I thought it was deemed that we just had to run the program five times, however I could see where the instructions would want the program to display five of each transformation on one pass. If this were the case, my program could easily loop through main five times in order to obtain five transformations, as all intermediate calculations are performed in helper functions. Another issue I have rarely encountered involves the built-in “resize()” function. Every once in a great while, this will fail for one of two reasons of which I could not determine the cause, and this function failure could not be avoided by a try-catch or if. If this were to happen one of the few times it seems to, the program can be run exactly the same again. That being said, I believe I did well in my implementation of this assignment.

In order to demonstrate my work, please find below sample images of the program running.

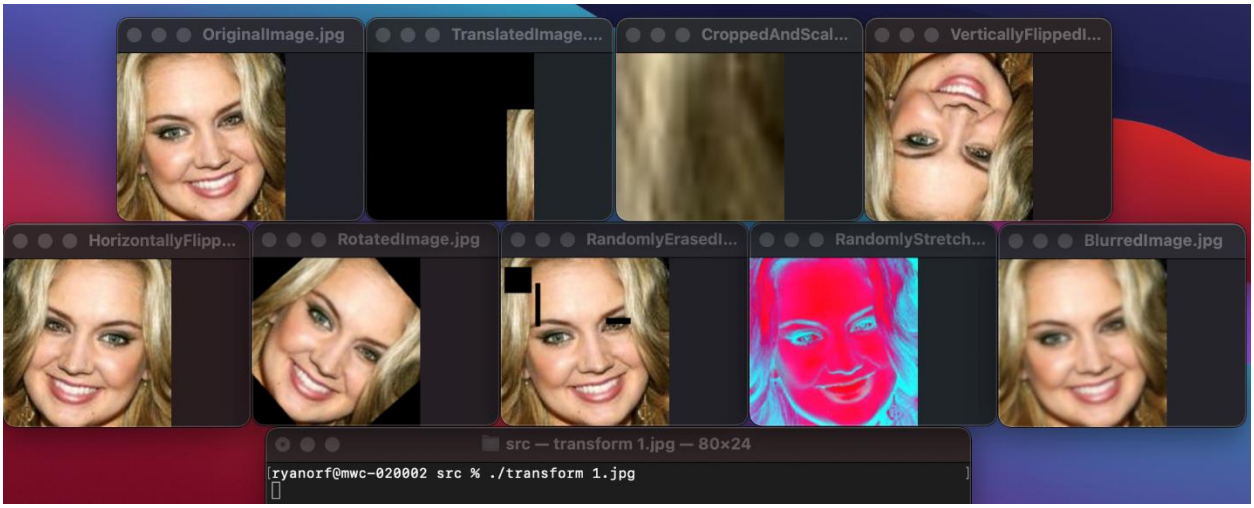
First Transformation of Image 1:



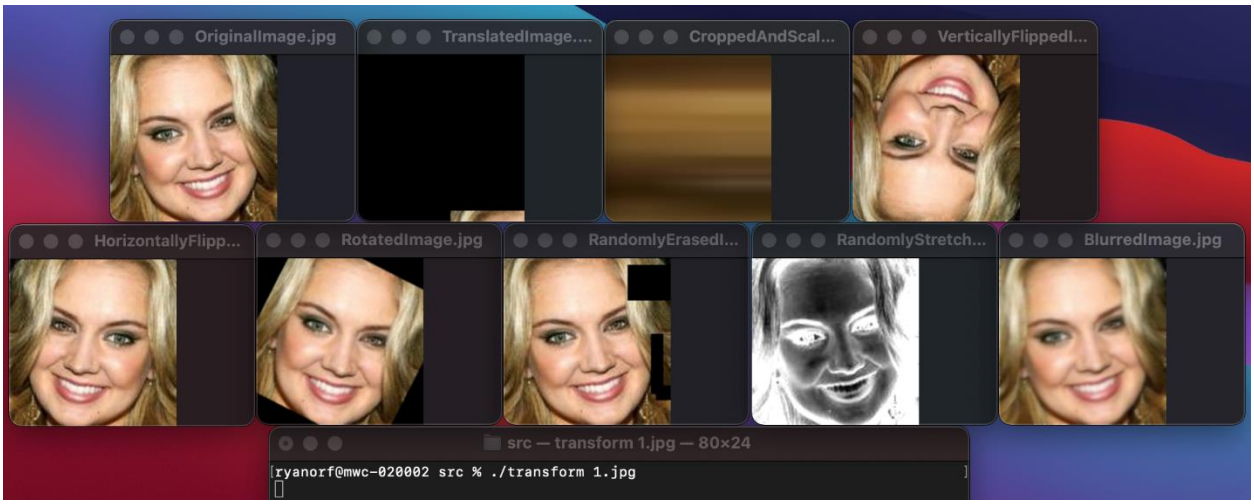
Second Transformation of Image 1:



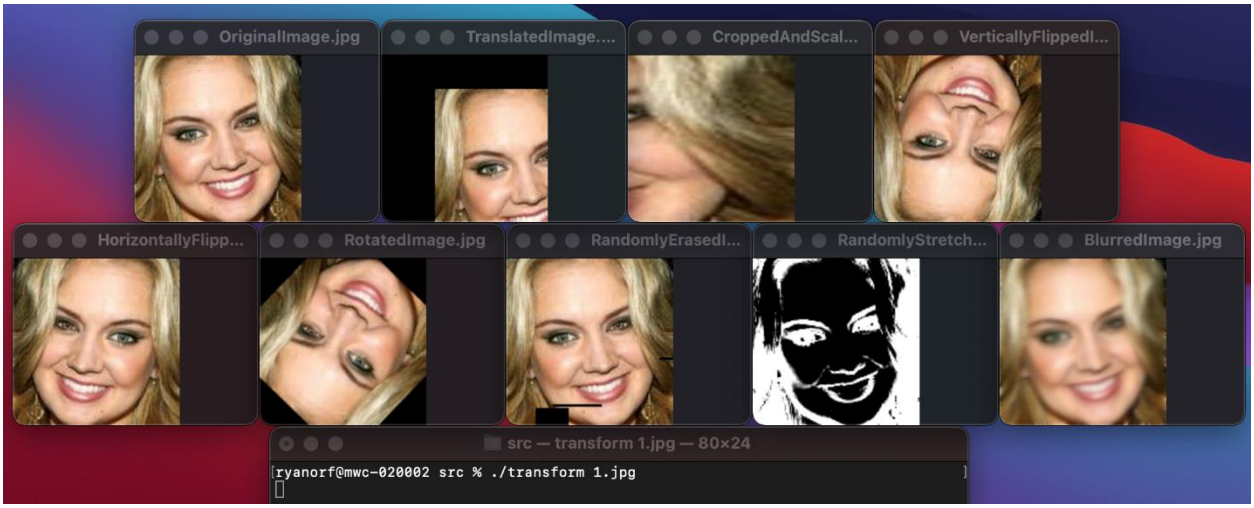
Third Transformation of Image 1:



Fourth Transformation of Image 1:



Fifth Transformation of Image 1:



Sample Transformation of Image 3 (to show the program works with different images of different sizes):