CS 4475 Project 1 Report

Sanghyun Chun

Shc1991@gatech.edu

902759995 (schun41)

Introduction

Having no background in photography whatsoever, I found it fascinating that I could actually write codes that would enable me to manipulate images and colors. I have no prior knowledge of Photoshop, and have only used very minimal Gimp until now.

I thought I would be interesting to mimic the seasonal changes using pixel manipulation. I would have an original image, and after running it through various filters, it would produce an image for each season. Below is how I achieved my result:

Workflow

Basic Overview

How to run the code (you can also refer to README.txt):

1. Have an image ready, and name the image "input.jpg". If what you have is a format other than jpg (e.g. PNG) you must convert it first into jpg.
2. Save input.jpg in the same folder as myproject1.py. Note: **you DO NOT need to change the size dimensions of your image**. Whatever the size of your input image is, the resulting images will retain the same size.
3. Run myproject1.py.

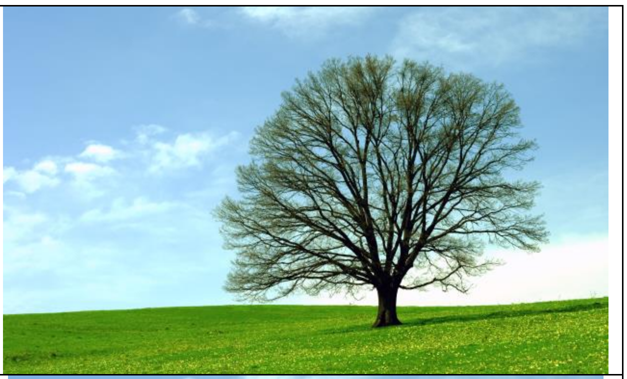
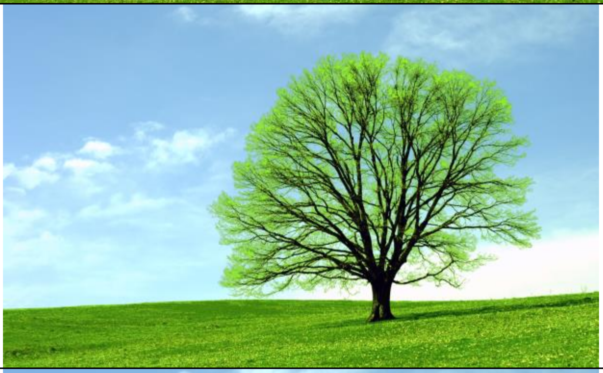
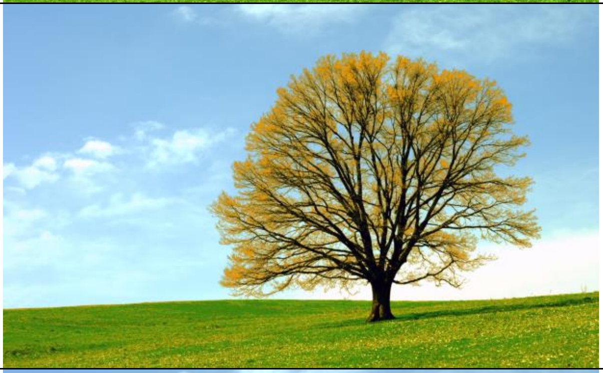
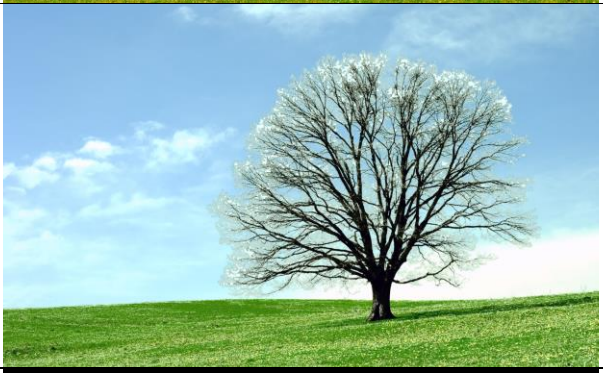
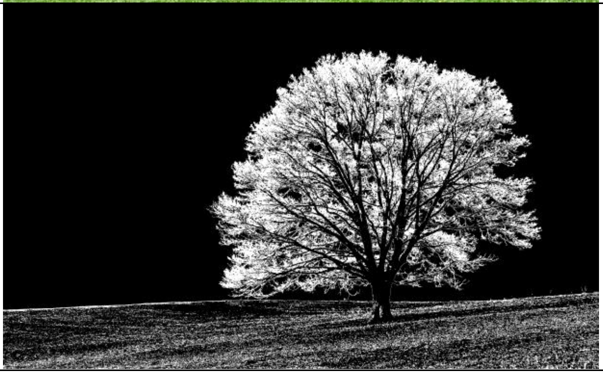
What you should see:

In the same folder as your input image, you should see 4 other images created:

1. imageMask.jpg
2. summerImage.jpg
3. fallImage.jpg
4. winterImage.jpg

Each image represents the mask image and the images that were produced after applying the seasonal filter functions.

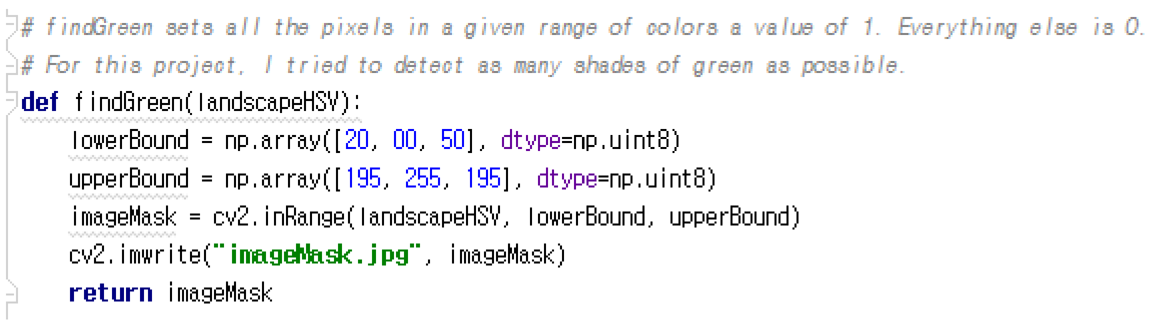
Below is an example of what you should get:

input.jpg	
summerImage.jpg	
fallImage.jpg	
winterImage.jpg	
imageMask.jpg	

In-Depth Analysis of Algorithms

To apply the filters, I needed to find a way to detect the color green in my input picture. However, most photos, if not all, has various shades of green, so simply using cv2.cvtColor was not possible. After researching on the OpenCV documentation, I came across **cv2.inRange**, which allowed me to set an upper bound and a lower bound RGB values. If a pixel falls between those ranges, it returns 1, otherwise a 0.

Using cv2.inRange, I now could make a function that would create a mask of the original image. I would first set an upper bound and lower bound, then run the input image through cv2.inRange. This would let me know where the green pixels are. I had to vigorously experiment with the upper and lower bounds to achieve maximal result. In the end, I found [20, 0, 50] and [195, 255, 195] a good compromise. Here is the code snippet for findGreen(image):

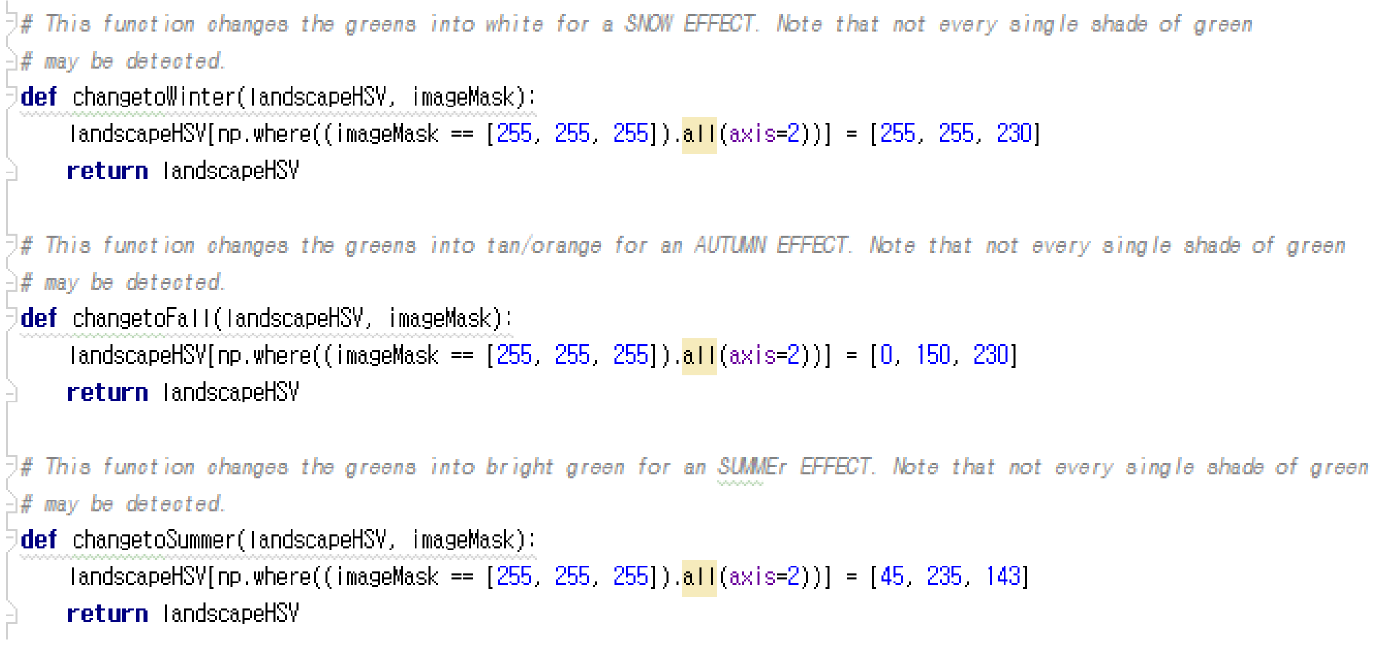


```python
# findGreen sets all the pixels in a given range of colors a value of 1. Everything else is 0.
# For this project, I tried to detect as many shades of green as possible.
def findGreen(landscapeHSV):
    lowerBound = np.array([20, 00, 50], dtype=np.uint8)
    upperBound = np.array([195, 255, 195], dtype=np.uint8)
    imageMask = cv2.inRange(landscapeHSV, lowerBound, upperBound)
    cv2.imwrite("imageMask.jpg", imageMask)
    return imageMask
```

For the filters themselves, I found **np.where** as the perfect solution. np.where allows you to return elements depending on the condition you set. So for my filters, I just needed to set a condition that if a value in imageMask is [255, 255, 255] (otherwise 1), change the corresponding pixel on input.jpg as [R, G, B]. As pseudocode, this would be:

```
Def filter(input, mask):
    Input[numpy.where(mask == [255, 255, 255])] -> change value to [R, G, B]
    Return input
```

Different filters would require a different RGB value. Again, I experimented with the RGB values multiple times. In the end, I settled with [45, 235, 143] for Summer, [0, 150, 230] for Fall, and [255, 255, 230] for Winter. The actual code can be seen here:



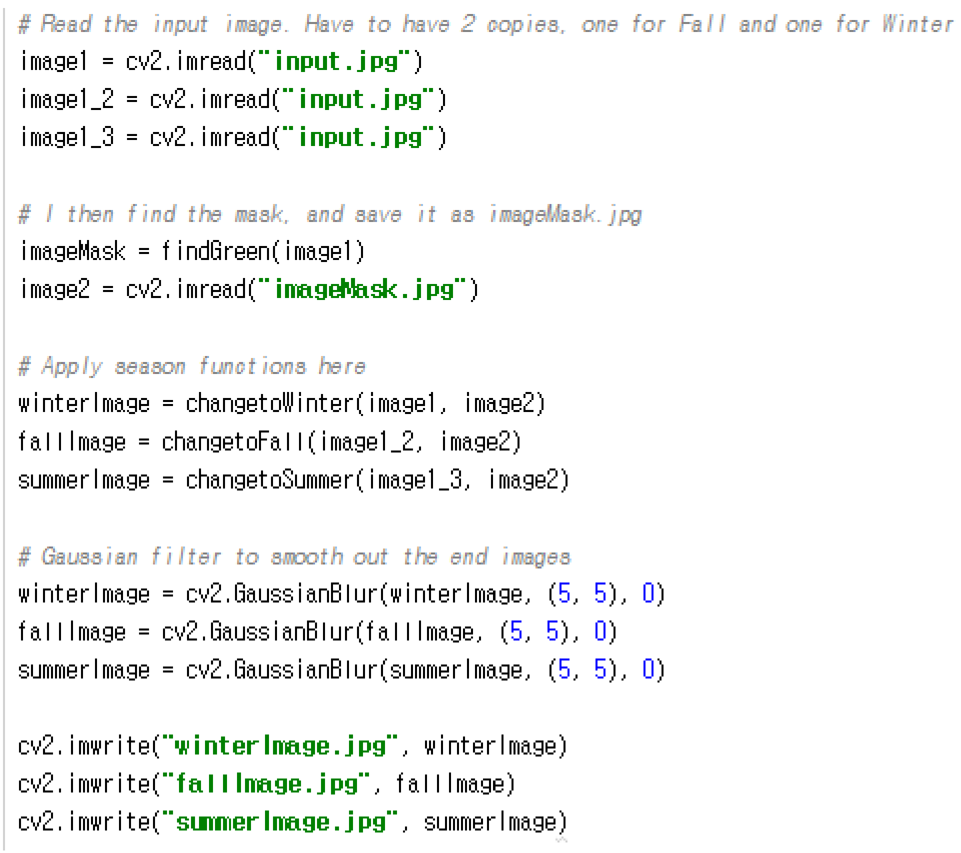
```python
# This function changes the greens into white for a SNOW EFFECT. Note that not every single shade of green
# may be detected.
def changetoWinter(landscapeHSV, imageMask):
    landscapeHSV[np.where((imageMask == [255, 255, 255]).all(axis=2))] = [255, 255, 230]
    return landscapeHSV

# This function changes the greens into tan/orange for an AUTUMN EFFECT. Note that not every single shade of green
# may be detected.
def changetoFall(landscapeHSV, imageMask):
    landscapeHSV[np.where((imageMask == [255, 255, 255]).all(axis=2))] = [0, 150, 230]
    return landscapeHSV

# This function changes the greens into bright green for an SUMMEr EFFECT. Note that not every single shade of green
# may be detected.
def changetoSummer(landscapeHSV, imageMask):
    landscapeHSV[np.where((imageMask == [255, 255, 255]).all(axis=2))] = [45, 235, 143]
    return landscapeHSV
```

Final Touches

Now that I’ve created functions for mask and filters, it was time to actually build the images. I needed to make 3 copies of the original image, one for each filter. This was necessary because after applying one filter, the image would have already been altered, making it unusable for the other filters. I saved each image after filtering as a variable, then applied GaussianFilter to smooth it out. I set the GaussianFilter level to 0 so that it did not blur out the final image too much. At the end, I saved each variable, giving them appropriate names. Here is the code snippet:



```
# Read the input image. Have to have 2 copies, one for Fall and one for Winter
image1 = cv2.imread("input.jpg")
image1_2 = cv2.imread("input.jpg")
image1_3 = cv2.imread("input.jpg")

# I then find the mask, and save it as imageMask.jpg
imageMask = findGreen(image1)
image2 = cv2.imread("imageMask.jpg")

# Apply season functions here
winterImage = changetoWinter(image1, image2)
fallImage = changetoFall(image1_2, image2)
summerImage = changetoSummer(image1_3, image2)

# Gaussian filter to smooth out the end images
winterImage = cv2.GaussianBlur(winterImage, (5, 5), 0)
fallImage = cv2.GaussianBlur(fallImage, (5, 5), 0)
summerImage = cv2.GaussianBlur(summerImage, (5, 5), 0)

cv2.imwrite("winterImage.jpg", winterImage)
cv2.imwrite("fallImage.jpg", fallImage)
cv2.imwrite("summerImage.jpg", summerImage)
```

Credits

For this assignment, I used 3 stock images from Google. Here are the links:

http://www.vbctulsa.com/wordpress/wp-content/uploads/2015/02/wrackthetree.jpg

http://kingofwallpapers.com/jungle/jungle-004.jpg

https://upload.wikimedia.org/wikipedia/commons/9/92/Christmas_tree_farm_IA.JPG

These were used for non-profit purposes.

For helps and references, the official OpenCV Documentation page and StackOverFlow were of tremendous help:

http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_tutorials.html

http://stackoverflow.com/

Artistic / Technical Emphasis

For this assignment, I leaned more towards the artistic side rather than the technical side. If you look at the code, it is not difficult at all. I guess the hardest part for me to write my codes was to find the appropriate functions (cv2.inRange and np.where). Everything else apart from these functions were all things that I've learned and practiced in-class (np.array, GaussianFilter). One technicality aspect of my project is that there is no need to change the size of the original image. This in essence allows the user to retain their photo's original format, and experience significantly less quality reduction (the only quality reduction would come from GaussianFilter). Overall, I feel like I showcased adequate amount of knowledge learned from the class. I would therefore allocate **20%** for technicality.

I feel much more confident about the artistic side of my project. As you may have noticed, my inRange filter does not detect every shade of green. There are many reasons to this. First, it is close to impossible to extract a whole range of green from a photo, since colors that do not seem green still contain a certain amount of green from RGB. Second, if I detect all shades of green and convert them to a given RGB, the resulting photo risks looking too blocky and unnatural. For instance, it is unlikely that in real life every blade of grass would be covered in snow; there needs to be a mix of white and green to look more natural. I believe the example table above showcases the artistic value of the end images. I would therefore allocate **30%** for artistic aspect of this project.