TED University

Department of Computer Engineering

CMPE 362

IMAGE PROCESSING HOMEWORK 1 REPORT

FATİH YİĞİTEL

38651154902

Brief explanation of what you have done

In the part 1

The aim is to combine two images - a background image and a foreground image - in a way that the foreground object is overlaying onto the background image.

The function is called combineForegroundBackground and takes in four parameters:

fgImg: The foreground image that we want to superimpose onto the background image

fgMask: A binary image that represents a mask of the foreground object, where the value is 255 at the pixels belonging to the foreground object and 0 at the remaining pixels.

bgImg: The background image onto which we want to superimpose the foreground object

topLeft: The coordinates of the top-left pixel of the foreground object in the output image.

In the Part 2

Using a binary mask that specifies the region of the foreground image to be combined, this task aims to combine a foreground image with a background image.

To put it another way, the purpose of the code is to use the mask to determine where to place the foreground image in order to composite it with the background image. The final image's quality is enhanced by reducing noise and enhancing edge details through the use of the Gaussian smoothing and unsharp masking steps.

The results that you have obtained together with the corresponding parameters and

discussion of how the results change as the parameters change

Part 1 code explanation:

This code imports the required libraries 'cv2' and 'numpy'. It defines a function named 'combineForegroundBackground', which accepts four arguments: the foreground image, the binary mask of the foreground object, the background image, and the position of the top-left corner of the foreground image on the background image.

To determine the overlapping region between the foreground and background images, it first retrieves the height and width of both images. It then calculates the intersection of the foreground image with the background image using the position of the top-left corner of the foreground image on the background image. Additionally, it computes the starting and ending coordinates of the intersection region in the foreground image.

Next, it creates a new copy of the background image. Finally, it replaces the pixels in the intersection region of the new background image with the corresponding pixels from the foreground image, using the binary mask of the foreground object to determine which pixels to keep. The function then returns the new background image with the inserted foreground object.

To obtain the results, the code loads the foreground image, background image, and binary mask of the foreground object from files. Then, it calls the 'combineForegroundBackground' function to insert the foreground object into the background image and display the outcome in a window.

Part 2 code explanation:

The necessary libraries, cv2 and numpy, are imported by this code.

We define the combineForegroundBackground function, which accepts the following four arguments: the background image, the foreground image's top-left corner on the background image, and the foreground object's binary mask on the foreground image.

To decide the locale of the frontal area picture that covers with the background picture we initially gets the level and width of the forefront and background pictures, and afterward computes the crossing point of the closer view picture with the background picture in light of the place of the upper left corner of the frontal area picture on the background picture. Additionally, we determine the intersection region's beginning and ending coordinates from the foreground image.

After that, a fresh copy of the background image is made.

Toward the end we supplant the pixels in the convergence district of the new background picture with the comparing pixels from the closer view picture, utilizing the double veil of the forefront object to figure out which pixels to keep and we returns the new background picture with the frontal area object embedded into it.

The foreground object's binary mask and the background image are loaded from files to produce the results. The foreground object is then inserted into the background image by calling the combineForegroundBackground function, and the result is displayed in a window.

Part 1 inputs, outputs, changes of the outputs according to changing of the parameters inputs.

Foreground İmage:

Mask Image

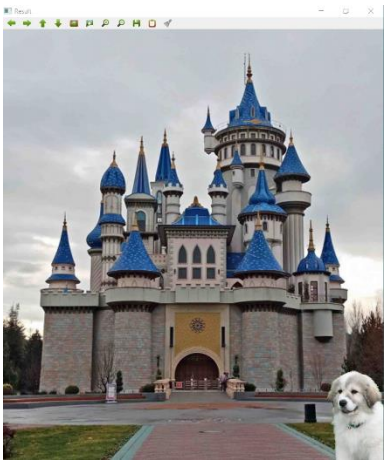
Background Image:

We use this inputs and we give the following x and y locations: (328,328)

And we get:

We use this inputs and we give the following x and y locations: (766, 656)

And we get:



We use this inputs and we give the following x and y locations: (300, 660)

And we get:

If we change the images it just means we use different images and the result with these images. However if we change the “topLeft” parameter (coordinates of the top-left pixel of the foreground object in the output image) the location of the foreground image will be changed. If borders are exceed. The exceeded parts of the foreground image not displayed:

As you see the dog’s foots are not displayed.

Part 2 inputs, outputs, changes of the outputs according to changing of the parameters inputs.

Foreground İmage:

Mask Image

Background Image:

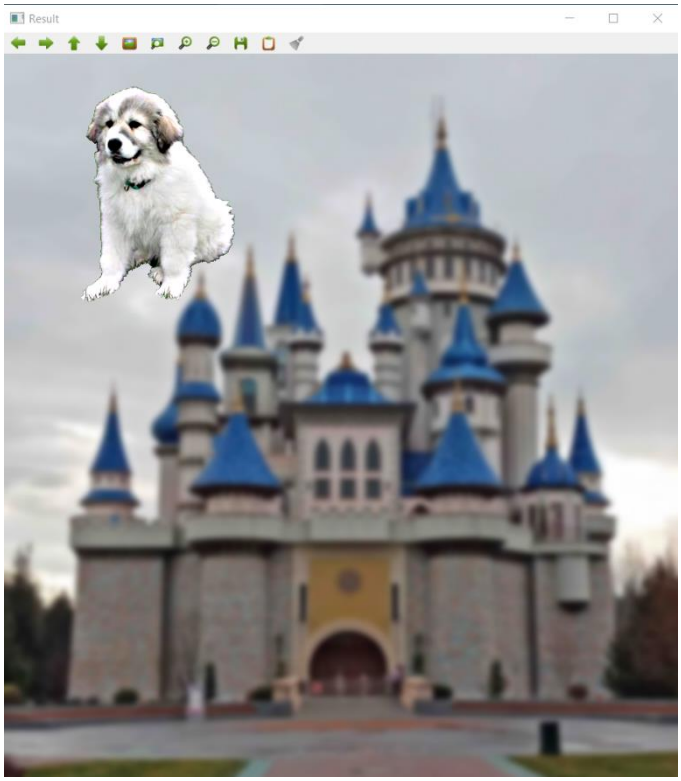
These are image inputs and we use another inputs that named topLeft, bgSigma and fgSigma.

In topLeft we define the place of the foreground image in the background image. So if we change it we see the different output image that the changing is only the location of the foreground image

However the fgSigma and bgSigma values for sharping the foreground image and smoothing the background image respectively. If we increase these values foreground image become more sharp and the fore ground image become more smooth

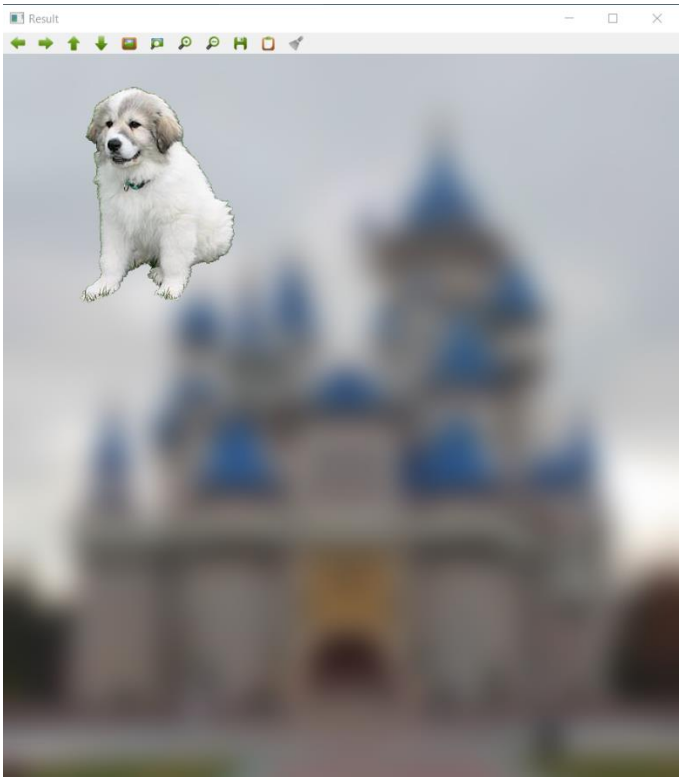
bgSigma :4

fgSigma : 10



bgSigma :20

fgSigma : 1



As you see when we increase bgSigma value the background image become smoother and when we decrease the fgSigma value the foreground image’s sharpness is decreased.

And to get normal images we do not use sigma values and the output: (bgSigma :0, fgSigma : 0)

Note*

I arrange the code to show the results in different window.