Akdeniz University

2018-2019, Spring

CSE366 Introduction to Image Processing

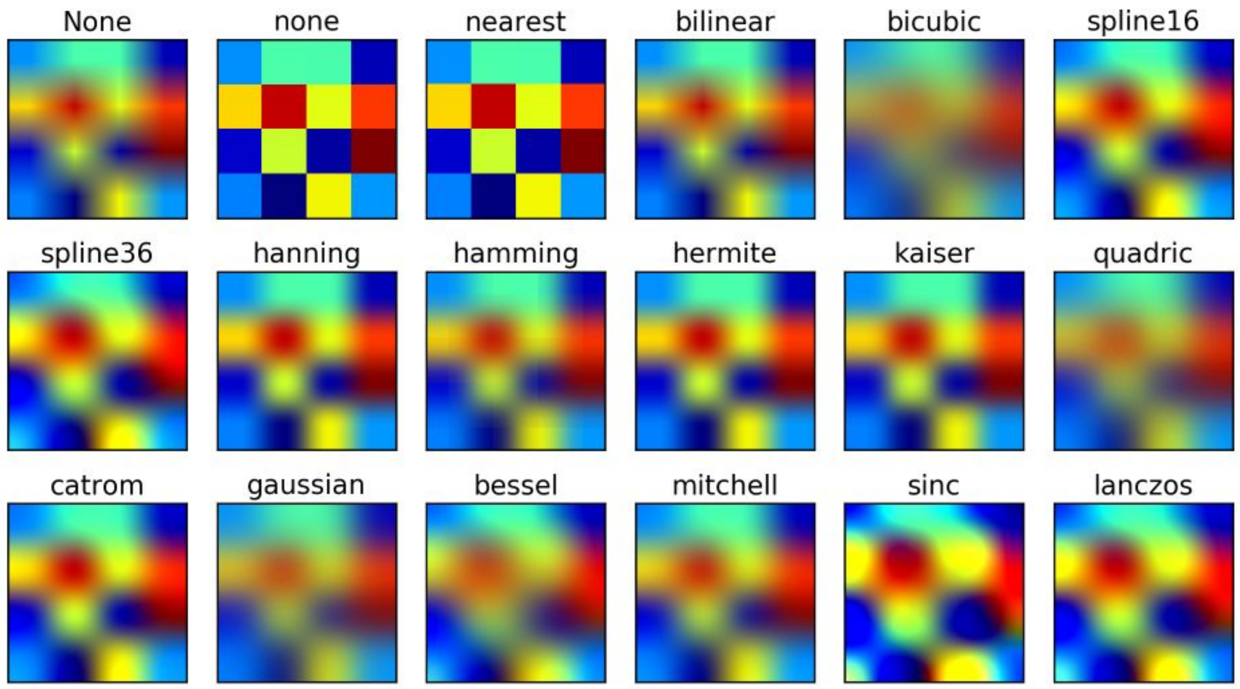
Homework 2- Size and Filtering

The following should be completed and submitted by the due date and time specified in google classroom homework instructions. Submissions received after the deadline will be subject to the late policy.

For homework 2:

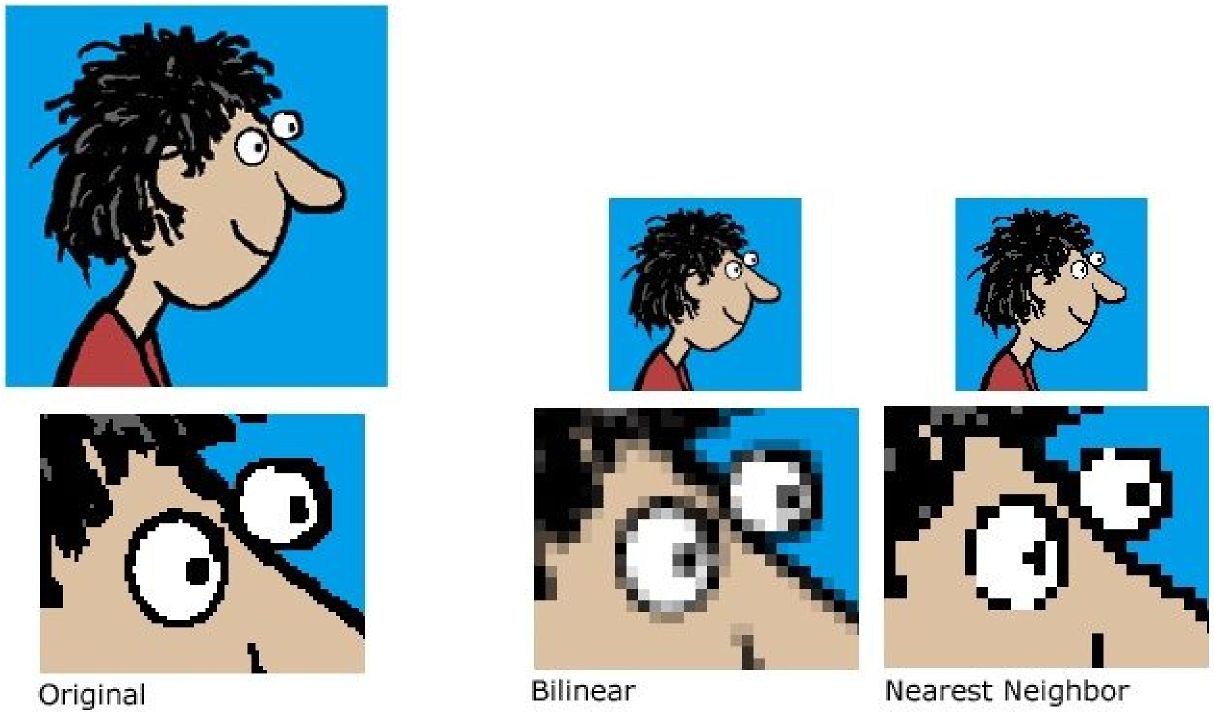
- You can use any programming language you prefer.
- You can use built in functions to load and save the images but for asked operations you need to do the work(you can use functions you made from the previous assignments).
- You can pick any image you like (it must be rgb).
- You can submit your work(in which your function outputs must be included) by google classroom or github.
- Example functions are just for an idea, you can constrict your functions the way you like.

There are multiple methods to resize an image, for this homework we will make functions that resize images using nearest neighbour and bilinear interpolations .

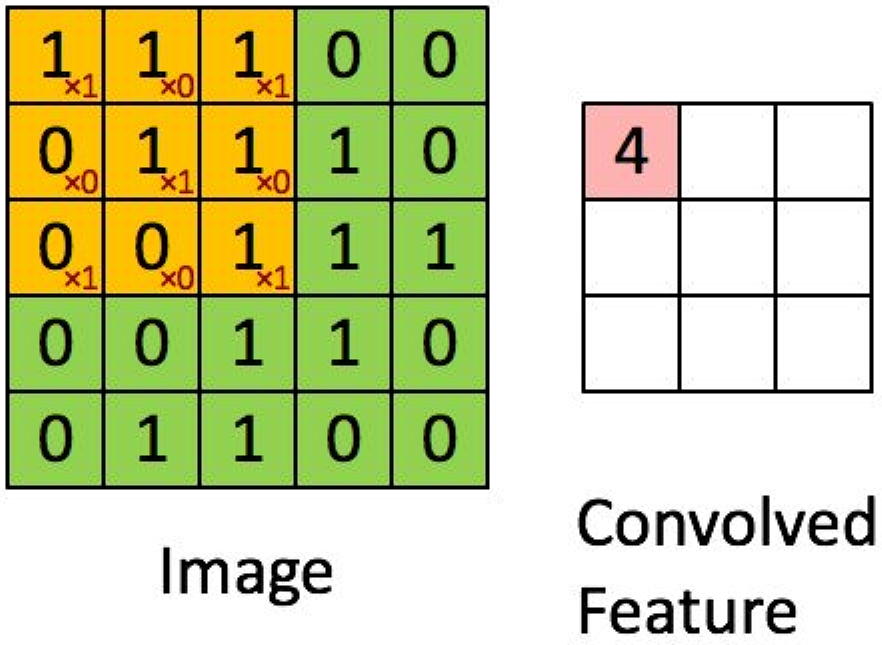


Pretty straightforward:

```
image neighbour_resize(image im, int w, int h)
{

}


image bilinear_resize(image im, int w, int h)
{

}
```



These functions will work to some extent but we need filters to polish results. Below gif shows convolutions. If you know the general idea behind convolutions you already figured what to do. Yes, we need to make convolution function and some filters to work with.



In order to try our convolve function we need to create a filter. Simplest filter to create is basic box filter we can use it for blurring. (Function below should create w x w image consist of 1’s.)

```
image boxfilt(int w)
{

}
```

There is one thing to be careful about. You can implement your filters the way you want but the images and the filters can have different number of channels. It would be impressing to implement a convolve function which is robust and works in every situation yet as long as it works you don’t have to do extra work.

```
image convolve(image im, image filter)

{


}
```

Now you can try your convolve function with the box filter you made. First filter your image and check the differences in between both pictures then downsize without filtering lastly downsize with filtering. You made something cool already!

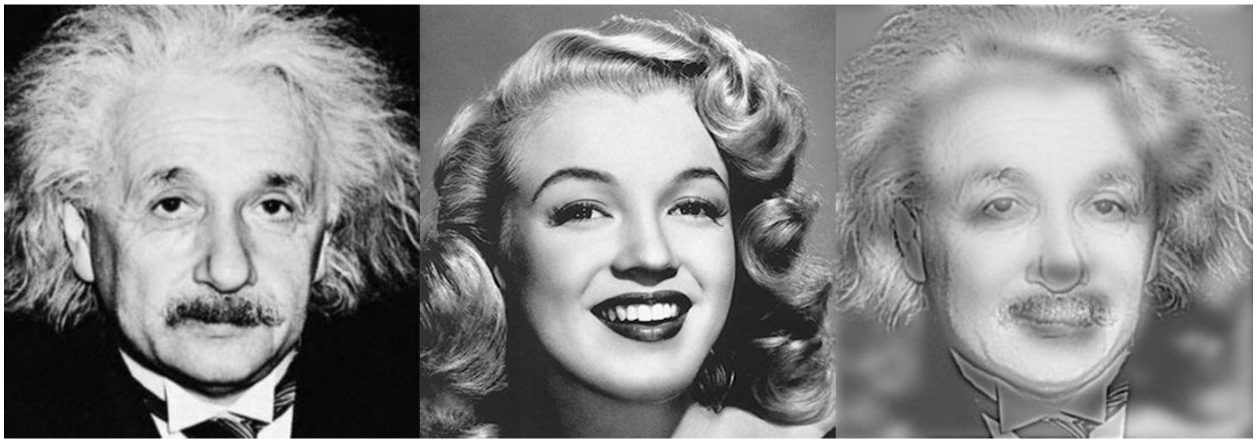
Make highpass, sharpen and emboss filters and check the results.

```
image  highpass_filter()

{


}


image  sharpen_filter()

{
}


image  emboss_filter()
{


}
```

Implement gaussian filter.(This one is harder than others but we need it)

```
image make_gaussian_filter(float sigma)
{
}
```

Gaussian filters are neat low pass filters using them we can separate low and high frequency of images.



You can guess what we are doing next: Hybrid images!!!

How to create hybrid images? If you like to read papers http://cvcl.mit.edu/hybrid/OlivaTorralb_Hybrid_Siggraph06.pdf

If you don’t like here is the summary:

-First pick two same sized images

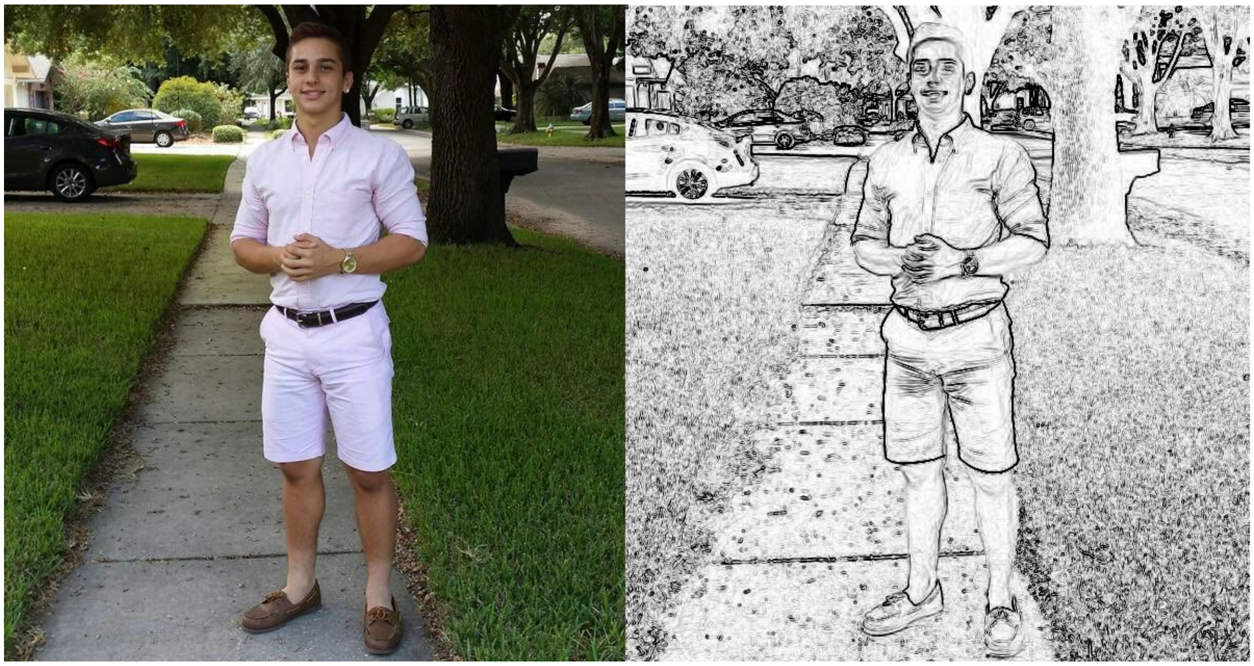
-Use gaussian filter on the first one and save it.

-Use gaussian filter on the second one and extract result from the original image.

-Now add them together.(I expect original results.)

```
image add_img(image a, image b)
{

}

image sub_img(image a, image b)
{

}
```

If you survived this far one last thing left to do Sobel filter! Edge detection is important part of image processing and computer vision implement sobel_filter to achieve this. (Note that a Sobel filter has two kernels, x-direction kernel and y-direction kernel. The x-direction kernel detects horizontal edges, and y-direction kernels detects vertical edges.)



```
image sobel_image(image im)

{


}
```