

The Hong Kong Polytechnic University

Spring 2022

COMP5523: Computer Vision and Image Processing

Project #2

Due: March 14, 2022

Welcome students,

This is a Python version of the second project of COMP5523, this project is slightly different from the C++ version. But the overall contents are same.

This project is also a code fill-in-blank task, we still have 11 test cases (**marked in red**) in **test_case.py** to evaluate your implementation. In order to make grading easier, you are only allowed to edit **resize_image.py** and **filter_image.py**. We use NumPy (1.19) and OpenCV-Python (4.5) to read and save images.

1. Image Resizing

The first 4 test cases are implementations of image resizing. To resize, we'll need some interpolation methods. We mainly focus on two types: (1) nearest-neighbor interpolation and (2) bilinear interpolation.

1.1 Nearest neighbor Interpolation

Test case 1 &2 rely on nearest interpolation.

Case 1. resize using nearest-neighbor interpolation with fixed 4x ratio.

Case 2. resize using nearest-neighbor interpolation with adaptive ratio.

Fill in **nearest_resize(img, w, h)** in **resize_image.py**.

You should:

- Create a new image that has same size as **img**.
- Loop over the pixels and map back to the corresponding coordinates.
- Use nearest neighbor interpolate to fill in the new image.

Your code will take the input image



The outputs for case1 and case2 should be:



And



1.2 Bilinear Interpolation

Test case 3 &4 rely on bilinear interpolation.

Case 3. resize using bilinear interpolation with fixed 4x ratio.

Case 4, resize using bilinear interpolation with adaptive ratio.

Fill in **bilinear_resize(img, w, h)** in **resize_image.py**.

The outputs are:



And

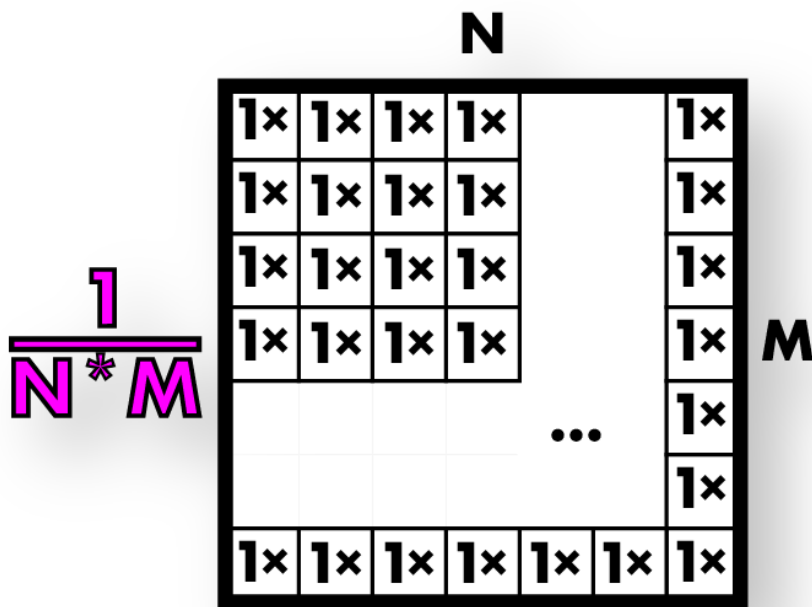


2. Image Filtering with Convolutions

We'll start out by filtering the image with a box filter. There are very fast ways of performing this operation but instead, we'll do the naive thing and implement it as a convolution because it will generalize to other filters as well.

2.1 Create Box Filter

We want to create a box filter, which as discussed in class looks like this:



Case 5, apply L1 normalization to the image.

One way to do this is make an image, fill it in with all ones, and then normalize it. That's what we'll do because the normalization function may be useful in the future.

- a) First fill in `l1_normalize(img)` in `filter_image.py`
- b) This should normalize an image to sum to 1.
- c) Next fill in `class make_box_filter` in `filter_image.py`. We will only use square filters `kernel_size` by `kernel_size`. It should be a square image with 1 channel with uniform entries that sum to 1.

Case 6, pad the image with zero.

Before implementing the convolutional kernel, one important thing is to deal with the border of the image. We can pad the image on the borders with zero, which is often used in CNNs.

Fill in `padding_image(img, filter)` in `filter_image.py`.

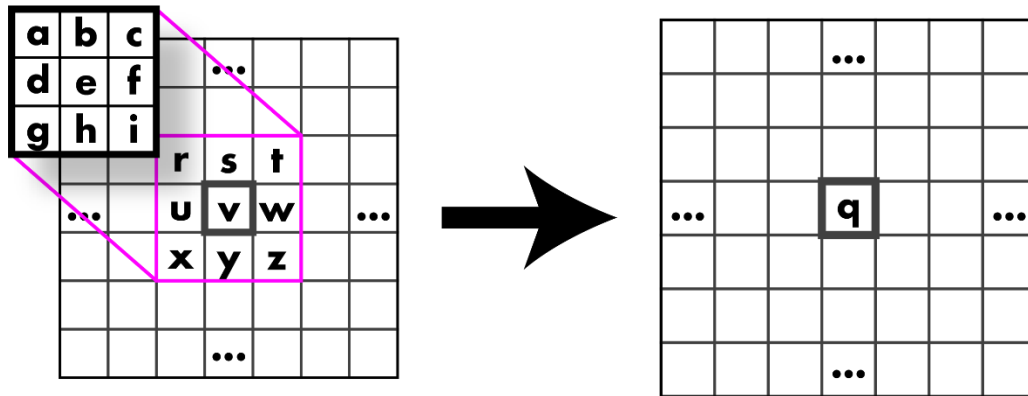
Here, we set a large filter size to highlight the border (in black). The output is:



2.2 Write a Convolution Function

Case 7, image convolution with 7x7 box filter, preserve is TRUE.

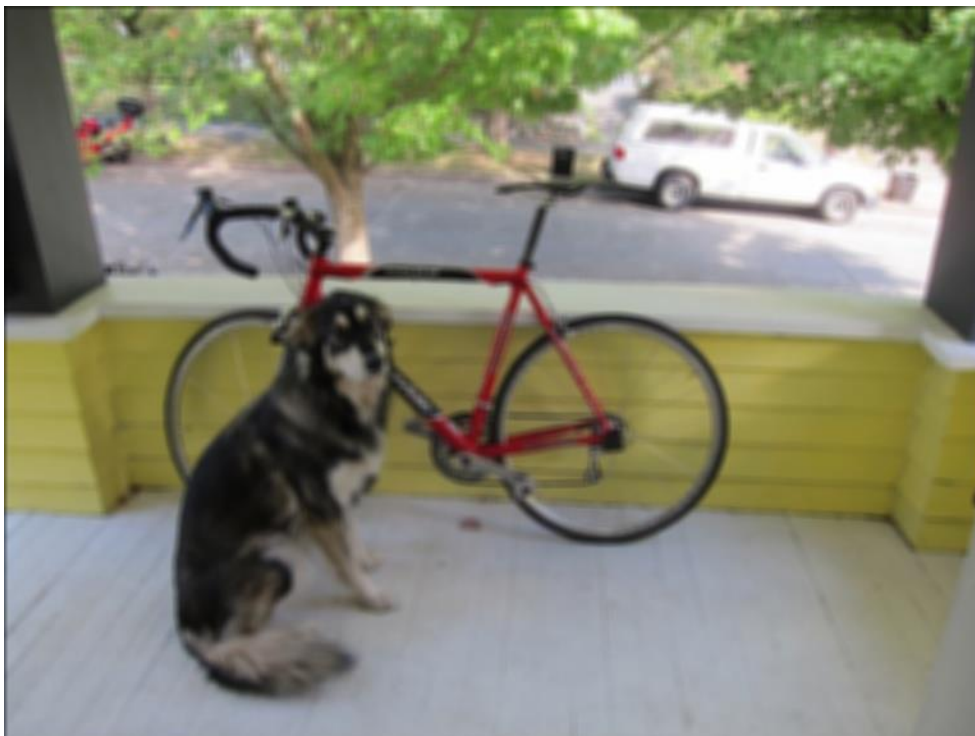
Then, we can discuss convolution. Just apply it to the image as we discussed in class:



$$q = a \times r + b \times s + c \times t + d \times u + e \times v + f \times w + g \times x + h \times y + i \times z$$

Fill in `convolve_image(img, filter, preserve)` in `filter_image.py`. For this function we have a few scenarios. With normal convolutions we do a weighted sum over an area of the image. With multiple channels in the input image there are a few possible cases we want to handle:

- If **preserve** is set to **True**, we should produce an image with the same number of channels as the input. This is useful if, for example, we want to run a box filter over an RGB image and get out an RGB image. This means **each channel** in the image will be filtered separately by the same filter kernel.
- If **preserve** is set to **False**, we should return a **1-channel** image, which is produced by applying the
- filter kernel to each channel, and then adding the channels together.
- The **filter** applied here should only have 1 channel. We check it.
- The output is a blur image and looks like this:



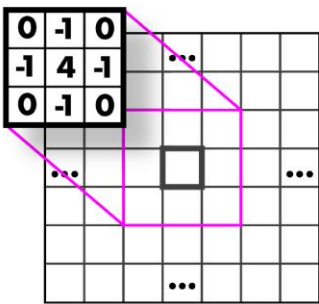
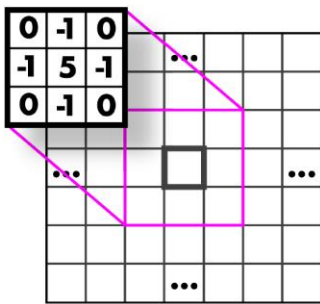
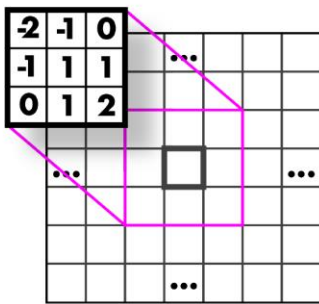



Case 8, image convolution with 7x7 box filter, preserve is FALSE.

If preserve set to False, the output should like this:



2.3 More Filters for Differnt Painting Styles

Here, we will try more filters: (1) 3x3 highpass filter, (2) 3x3 sharpen filter and (3) 3x3 emboss filter:

Highpass	Sharpen	Emboss
		
		

Case 9, image convolution with 3x3 highpass filter, preserve is FALSE

Fill in **class make_highpass_filter** in **filter_image.py**

The output should look like:



Case 10, image convolution with 3x3 sharpen filter, preserve is TRUE

Fill in **class make_sharpen_filter** in **filter_image.py**

The output should look like:



Case 11, image convolution with 3x3 emboss filter, preserve is TRUE

Fill in class `make_emboss_filter` in `filter_image.py`

The output should look like:



Grading for 11 Test Cases

- a) The grading is same with C++ version, you test all the cases in `tes_case.py`.
- b) You are only allowed to edit `resize_image.py` and `filter_image.py`. And 11 test cases are in `test_case.py`.
- c) Test case 1&5 are 5 scores, others are 10 scores. Totally, 100 scores.

Important Note: Please carry out your project independently! Your answer should NOT be identical or significantly similar to the answers from any of your classmates. Once your answer is found suspicious, all students with similar answers will be subject to investigation of academic dishonesty and may receive penalty for any misconducts.