The Hong Kong Polytechnic University

Spring 2022

COMP5523: Computer Vision and Image Processing

Project #2

Due: March 14, 2022

Welcome students,

It's time for project 2. This one may be a little harder than the last one so remember to start early and save often!

In order to make grading easier, please only edit the files we mention. You should be able to submit `resize_image.cpp` and `filter_image.cpp`, and we should be able to compile and evaluate them by running the `test_case` excutable file.

Do not change any of the other files and do not change the signature lines of the functions.

This handout contains 11 test cases to help you complete this project and evaluate your implemenation. These cases (need your coding to complete the function) are highlighted in **red**. The programming language is C++.

1. Image Resizing

We've been talking a lot about resizing and interpolation in class, now's your time to do it! To resize we'll need some interpolation methods and a function to create a new image and fill it in with our interpolation methods. We mainly focus on two types: (1) nearest-neighbor interpolation and (2) bilinear interpolation.

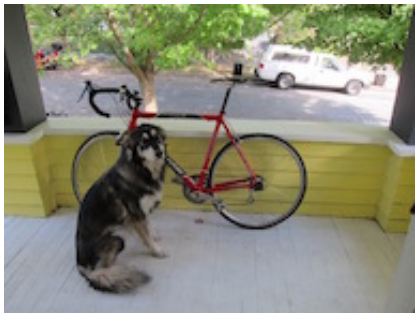
1.1 Nearest-neighbor Interpolation

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

- Fill in `Image nearest_resize(const Image& im, int w, int h)` in `src/resize_image.cpp`.
- It should:
    - Create a new image that is `w x h` and the same number of channels as `im`.

    - 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 output is:

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

- Do nearest-neighbor interpolation but with adpative scaling ratio on the image.
- Fill in `Image nearest_resize_adaptive(const Image& im, int w, int h)` in

`src/resize_image.cpp`.

- The output is:

1.2 Bilinear Interpolation

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

- Now, we change the method by using bilinear interpolation for better recovery.
- Fill in `Image bilinear_resize(const Image& im, int w, int h)` in `src/resize_image.cpp`.
- The output is:

Case 4, resize using bilinear interpolation with adaptive ratio

- Similar, we do the bilinear interpolatio but with adpative scaling ratio on the image.
- Fill in `Image bilinear_resize_adaptive(const Image& im, int w, int h)` in `src/resize_image.cpp`.
- The output is:

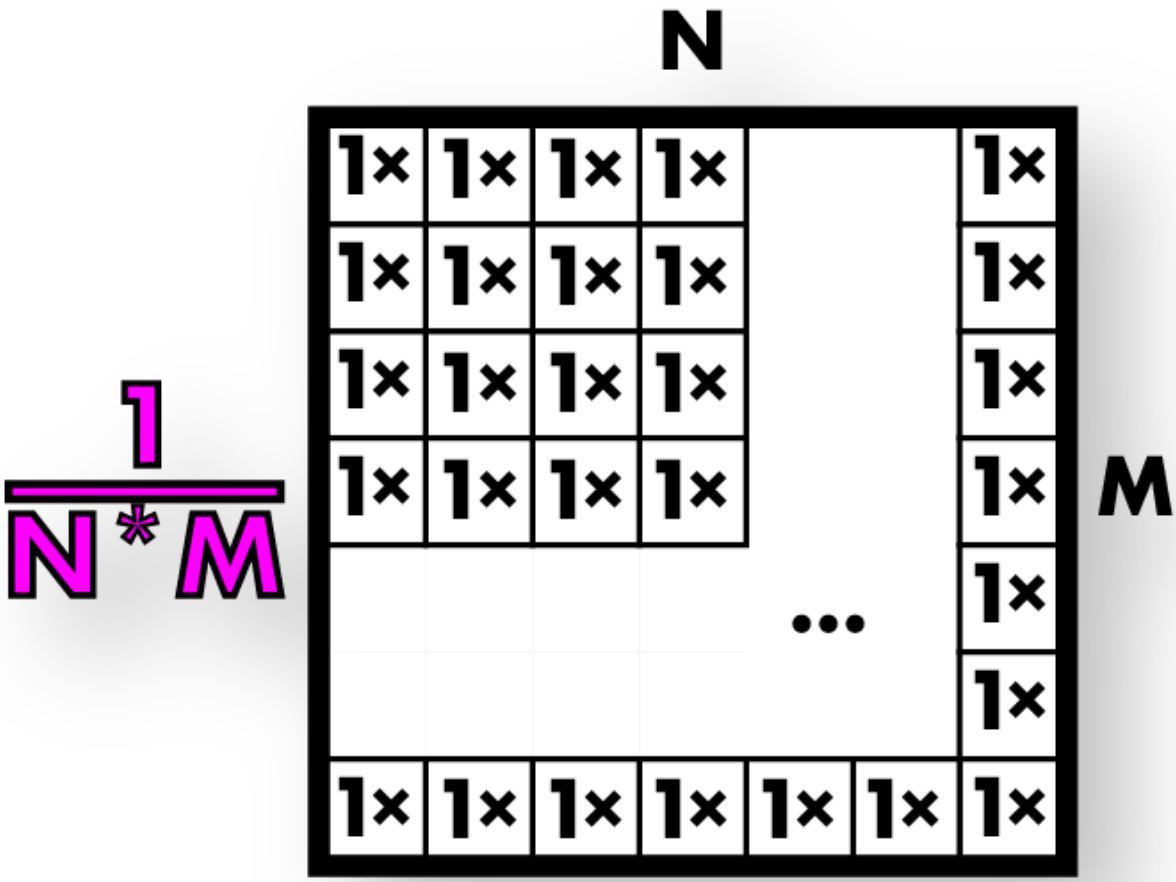
Next, we can construct filters to extract interesting features (e.g., texture and edge) to shift the painting style of the entire image. This is fun!

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!

- First fill in `mage l1_normalize(const Image& im)` in `filter_image.cpp`.
- This should normalize an image to sum to 1.
- Next fill in `Image make_box_filter(int w)` in `filter_image.cpp`. We will only use square box filters so just make your filter `w x w`. 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.

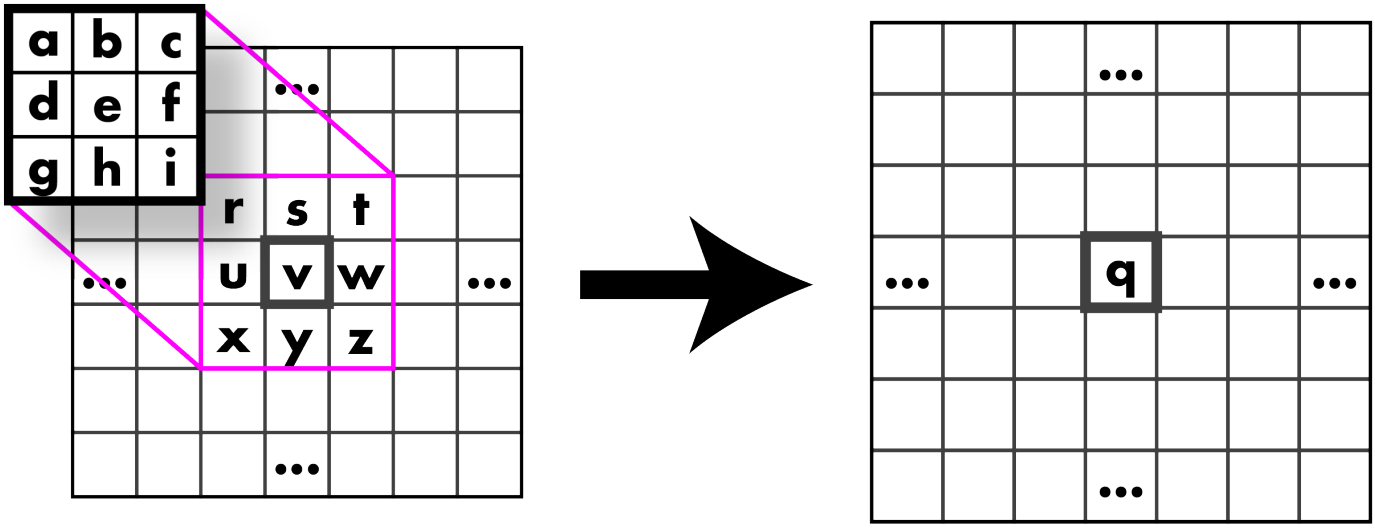
- Fill in `Image padding_image(const Image& im, const Image& filter)` in `src/filter_image.cpp`.

- 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. **We are calling this a convolution but you don't need to flip the filter or anything (we're actually doing a cross-correlation)**. Just apply it to the image as we discussed in class:



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

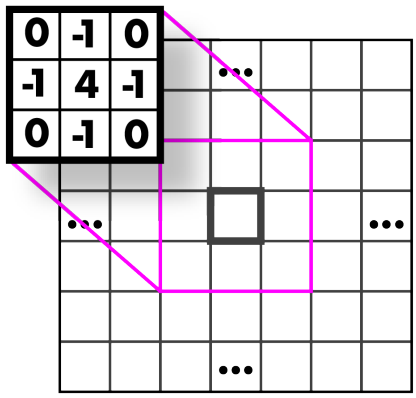
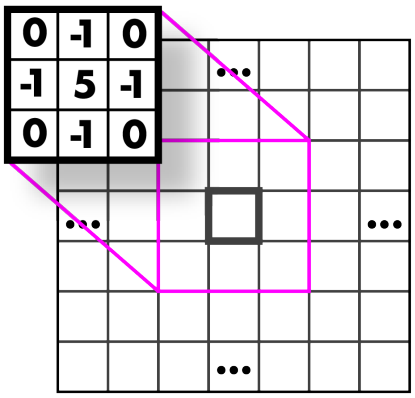
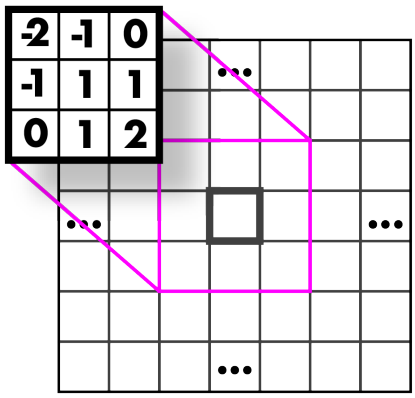
- Fill in `Image convolve_image(const Image& im, const Image& filter, bool preserve)` in `src/filter_image.cpp`. For this function we have a few scenarios. With normal convolutions we do a weighted sum over an area of the image. With multiple channels in the input image there are a few possible cases 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. UNLESS:
- 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 this with an `assert`.
- The output is a blur image and looks like this:

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

- We also check the results when `preserve` is `false`.
- The output looks like:

2.3 More Filters for Different Painting Styles

We can change the construction of the filter to extract different features and shift the image into other painting styles. Here, we will try three types: (1) 3x3 highpass filter, (2) 3x3 sharpen filter and (3) 3x3 emboss filter. Here, we give a comparison on them:

Highpass	Sharpen	Emboss
		
		

Let's try these filters!

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

- Fill in the functions `Image make_highpass_filter()` in `src/filter_image.cpp`
- The output looks like:

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

- Fill in the functions `Image make_sharpen_filter()` in `src/filter_image.cpp`
- The output looks like:

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

- Fill in the functions `Image make_emboss_filter()` in `src/filter_image.cpp`
- The output looks like:

Grading for Test Cases

- There are 11 test cases to check your implementation. -You can build `test/test_case.cpp` and run the executable `test_case` to make a simple check to your implementation. All the modified images (after resizing and filtering) are save in `output/test_case_output` .
- You only need to modify and submit **ONLY 2 files:** **`resize_image.cpp`** **and** **`filter_image.cpp`** .
- Do not change any of the other files and do not change the signature lines of the functions. We will rely on those to be consistent for testing. You are free to define any amount of extra structs/classes, functions, global variables that you wish.

Grading

```
// Case 1, resize using nearest neighbor interpolation with fixed 4x ratio,   5 cr
edits
test_nn_resize_4x();

// Case 2, resize using nearest neighbor interpolation with adaptive ratio,  10 cr
edits
test_nn_resize_adaptive();

// Case 3,  resize using bilinear interpolation with fixed 4x ratio,         10 cr
edits
test_bl_resize_4x();

// Case 4, resize using bilinear interpolation with adaptive ratio,          10 cr
edits
test_bl_resize_adaptive();

//Case 5, apply l1 normalization to the image,                               5 cr
edits
test_l1_normalization();

// Case 6, pad the image with zero,                                          10 cr
edits
test_padding_image();

// Case 7, image convolution with 7x7 box filter, preserve is TRUE,          10 cr
edits
test_convolution();

// Case 8, image convolution with 7x7 box filter, preserve is FALSE,         10 cr
edits
test_convolution_no_preserve();

// Case 9, image convolution with 3x3 highpass filter, preserve is FALSE,    10 cr
edits
test_highpass_filter();

// Case 10, image convolution with 3x3 sharpen filter, preserve is TRUE,     10 cr
edits
test_sharpen_filter();

// Case 11, image convolution with 3x3 emboss filter, preserve is TRUE,      10 cr
edits
test_emboss_filter();

Total: 100 credits
```

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.