Homework 2: More Image Processing with Higher-Order Functions

Due Wednesday, September 21, 2022 at 11:59pm

Introduction

Following up to Homework 1, in Homework 2 you will perform more processing tasks with higher order functions. As before, we use the types `Pixel` and `Image` to specify our functions:

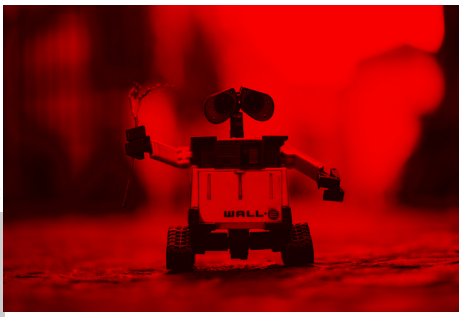
1. A **Pixel** is a three-element array, where each element is a number in the range 0.0 to 1.0 inclusive.
2. An **Image** is an object whose 2D array of Pixels is accessed via gePixel / setPixel.

Programming Task

1. Write a function called `imageMapCoord` with the following type:

```
imageMapCoord(img: Image, func: (img: Image, x: number, y: number) => Pixel): Image
```

The result must be a new image with the same dimensions as `img`. For each pixel in the new image, its value should be the result of applying `func` to the corresponding pixel of `img`.
This function is more general than `imageMap`: the new pixel value may also depend on the coordinates of the original pixel. On the right you see a sample output of using `imageMapCoord` as follows:



```
let url =
'https://people.cs.umass.edu/~joydeepb/robot.jpg';
let robot = lib220.loadImageFromURL(url);
imageMapCoord(robot, function(img, x, y) {
  return [img.getPixel(x, y)[0], 0, 0];
}).show();
```

2. Write a function called `imageMapIf` with the following type:

```
imageMapIf(img: Image, cond: (img: Image, x: number, y: number) => boolean,
                       func: (p: Pixel) => Pixel): Image
```

The result is a new image. In it, the value of the pixel at *(x, y)* is either (a) the value `func(p)`, where `p` is the original pixel, when `cond(img, x, y)` returns `true`, or (b) identical to the original pixel otherwise. **You may not use loops in this function. Instead, use imageMapCoord defined above.**

3. Write a function called `mapWindow` with the following type:

```
mapWindow(img: Image, xmin: number, ymin: number, xmax: number, ymax: number,
                      func: (p: Pixel) => Pixel): Image
```

The result is a new image. In it, the value of the pixel at *(x, y)* is either (a) the value `func(p)`, where `p` is the original pixel, if the pixel coordinates *(x, y)* are in the interval [xmin, xmax] and [ymin, ymax] respectively, or (b) identical to the original pixel otherwise. Use `imageMapIf`.

For the following functions, you may not use loops. Instead, use one of the higher-order functions defined above or in Homework 1. Carefully selecting which function to use will allow you to write your code more concisely and promote reuse, which are important points of the assignment.

4. Write a function called `makeBorder` with the following type:
```
makeBorder(img: Image, thickness: number, func: (p: Pixel) => Pixel): Image
```
The result is a new image, where each pixel whose distance to some edge pixel of the image is less than `thickness` has the value `func(p)`, where `p` is the original pixel. Other pixel values are unchanged. The distance between two pixels at (x1, y1) and (x2, y2) is defined as $|x1-x2|+|y1-y2|$.

5. Write a function called `dimCenter` with the following type:
```
dimCenter(img: Image, thickness: number): Image
```
The result is a new image, where a border of `thickness` pixels along the image edges are unchanged from the input. (A pixel is in the border if it has distance less than `thickness` to some edge pixel). The remaining pixels have each color value (intensity) reduced by 20%. Use a suitable function defined above.

6. Write a function called `redBorder` with the following type:
```
redBorder(img: Image, thickness: number): Image
```
The result is a new same-size image with a red border of `thickness` pixels along each edge of the image. Other pixel values are unchanged. Use a suitable function defined above.

7. Write a function called `grayBorder` with the following type:
```
grayBorder(img: Image, thickness: number): Image
```
The result is a new same-size image with a grayscale border of `thickness` pixels along each edge of the image: if the input pixel value is (r, g, b), its value in the output will be (m, m, m) where m is the average of r, g, and b. Other pixel values are unchanged. Use a suitable function defined above.

Note: your functions must not produce any run-time errors, irrespective of the input image dimensions.

Testing Your Code

As you know, an important part of a project is testing your code thoroughly. In this project, in addition to testing with a variety of input images, you should also use a variety of input *functions* to test your higher-order functions for correctness. To get you started, we have provided a few test cases here. You should define your own additional tests, which will count towards the grade.

- The value returned by imageMapCoord is an image, and must be distinct from the input image.

```
test('imageMapCoord function definition is correct', function() {
  function identity(image, x, y) { return image.getPixel(x, y); }
  let inputImage = lib220.createImage(10, 10, [0, 0, 0]);
  let outputImage = imageMapCoord(inputImage, identity);
  let p = outputImage.getPixel(0, 0); // output should be an image, getPixel works
  assert(p.every(c => c === 0));      // every pixel channel is 0
  assert(inputImage !== outputImage); // output should be a different image object
});
```

- Test an identity function with imageMapCoord. The resulting image should be unchanged. For this test, we will re-use the pixel equality testing helper function from project 1.

```
function pixelEq (p1, p2) {
  const epsilon = 0.002; // increase for repeated storing & rounding
  return [0,1,2].every(i => Math.abs(p1[i] - p2[i]) <= epsilon);
};
```

```
test('identity function with imageMapCoord', function() {
  let identityFunction = function(image, x, y ) {
    return image.getPixel(x, y);
  };
  let inputImage = lib220.createImage(10, 10, [0.2, 0.2, 0.2]);
  inputImage.setPixel(0, 0, [0.5, 0.5, 0.5]);
  inputImage.setPixel(5, 5, [0.1, 0.2, 0.3]);
  inputImage.setPixel(2, 8, [0.9, 0.7, 0.8]);
  let outputImage = imageMapCoord(inputImage, identityFunction);
  assert(pixelEq(outputImage.getPixel(0, 0), [0.5, 0.5, 0.5]));
  assert(pixelEq(outputImage.getPixel(5, 5), [0.1, 0.2, 0.3]));
  assert(pixelEq(outputImage.getPixel(2, 8), [0.9, 0.7, 0.8]));
  assert(pixelEq(outputImage.getPixel(9, 9), [0.2, 0.2, 0.2]));
});
```