LAB 6

Dealing with digital images in MATLAB

MATLAB has a toolbox (i.e., library) that is useful for dealing with digital images. The Image Processing Toolbox contains many functions that can be used to read in, modify, and write images in different formats. The functions needed for this problem are:

- `imread(filename, format)` - reads in a digital image file with the given `filename` in the specified `format`. Both `filename` and `format` are strings and for this assignment, the format will always be `'jpg'`. A matrix representation of the image is returned. Example:
  - `filename = '….jpg';`
  - `image = imread(filename, 'jpg');`
- `imshow(image)` - displays the image in a separate image window. Example:
  - `imshow(image)`

The main program

The main program (which has been partially completed for you) first reads in the digital image file ….jpg (bmp) and displays it. The program then repeatedly presents the user with a menu of choices (numbered from 1 to 6), prompts to enter a choice and reads it in, and performs the desired actions, until the user enters `0` to quit. The menu options are as follows:

1. `new image` - prompts the user to enter the name of jpg image file, reads in the file and makes the image the new current image, and displays the new current image. You may assume the user always enters the name of a file which can be read in as a jpg image.
2. `original` - displays the current image in its original (un-filtered) state
3. `negative` - applies the `negative` filter to the original current image and displays the filtered image
4. `sharpen` - applies the `sharpen` filter to the original current image and displays the filtered image
5. `posterize` - applies the `posterize` filter to the original current image and displays the filtered image
6. `blur` - applies the `blur` filter to the original current image and displays the filtered image

If the user enters in something other than an integer between 0 and 6 (inclusive), the program displays `invalid choice`, redisplays the menu, and prompts the user for a new choice.

The image filters

Each image filter is a function which takes one argument, the image matrix. The function returns a new image matrix which results from applying the desired filter to the image given in the argument. The filters all have the same basic algorithm:

```
for each row r in the image matrix
```

```
    for each column c in row r in the image matrix
        assign a value to the pixel at location (r, c) in the
new image based on the filter
```

The `negative` filter

The `negative` filter creates a "negative" image: black pixels become white and white pixels become black. To achieve this effect, each pixel's value is subtracted from 255.

The `sharpen` filter

The `sharpen` filter makes an image "sharper" by make dark pixels darker and light pixels lighter. More precisely, the pixel values in the new image are determined as follows: if the pixel value in the original image is less than 128, the pixel value in the new image is obtained by subtracting 16 from the pixel value in the original image; otherwise, the pixel value in the new image is obtained by adding 16 to the pixel value in the original image.

The `posterize` filter

The `posterize` filter creates a poster-like effect by reducing the number of possible pixel values to 4 (i.e., black, white, light gray, and dark gray) using the following conversion table:

original pixel value	new pixel value
0 - 71	0
72 - 127	72
128 - 183	183
184 - 255	255

The `blur` filter

The `blur` filter creates a blurred image. A pixel value in the new image is obtained by taking an average (mean) of the pixels around the original image. More specifically, to find the pixel value at row r, column c in the blurred image, find the average of the pixel values in the original image in the 5 x 5 square matrix centered on the pixel at (r, c), rounded to the nearest integer. Note that around the edges of the matrix, it will not be possible to have an entire 5 x 5 square centered on a given pixel; for those pixels, you will find the average of as much of the 5 x 5 square as exists within the image matrix.

For example, suppose our image has the following pixel matrix:

```
 10    10    80    80   200   200
 10    10    80    80   200   200
 30    30   120   120   220   220
```

```
30    30    120   120   220   220
50    50    160   160   240   240
50    50    160   160   240   240
```

Then, the value of the pixel in the blurred image at row 3, column 4 would be 136, which is computed by taking the average of the pixels shown in **bold red** below:

```
10    10     80    80   200   200
10    10     80    80   200   200
30    30    120   120   220   220
30    30    120   120   220   220
50    50    160   160   240   240
50    50    160   160   240   240
```

and the value of the pixel in the blurred image at row 1, column 2 would be 55, which is computed by taking the average of the pixels shown in **bold blue** below:

```
10    10     80    80   200   200
10    10     80    80   200   200
30    30    120   120   220   220
30    30    120   120   220   220
50    50    160   160   240   240
50    50    160   160   240   240
```

Hints:

- To implement the calculation of the average, think about creating a doubly-nested `for`-loop to iterate over a 5x5 square centered a pixel (r, c) (without worrying about whether the entire square is inside the image or not). Then, inside the body of the nested `for`-loops, check to see if the location being considered in that iteration is inside the image matrix; if it is, use that pixel in the calculation of the average (and if it isn't, just ignore that location).
- Remember that the average = (sum of pixel values)/(number of values in sum). Since not every square is entirely within the bounds of the image matrix, the number of values in the sum will not always be 25.

The mirror filter

The `mirror` filter creates a mirror image of the original image. You can get up to 3 points in extra credit by writing a `mirror` filter and adding a menu option `7` to your main program.