Problem 1A: A quick practice run on working with 2D images

Pertinent readings for Problem 1:

1. Graphical intuition for 2D convolutions

(Link also on Blackboard, under /Readings & Thoughts / Lecture [6]

https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1

2. Different kinds of 2D filters

(check out the links on Blackboard (near the bottom of that page): /Readings & Thoughts / Lecture [6])

3. How to calculate 2D convolutions

   a) Blackboard: /Readings & Thoughts / Lecture [6] -> 2 links near the bottom of that page)

   b) Blackboard: /Lectures / Lecture [6] -> 1 giant example at the bottom of that page
   (we briefly went over this during recitations on Fri 9/27/19)

4. Rangayyan (Biomedical image analysis, 1st ed):

(Download from Blackboard, under /Resources / Signal processing texts)

Ch. 4:	pages 219	(The 2D “flip & slide” method for convolutions)
	pages 314 – 323	(Applications: Unsharp masks / Laplacian subtractions)
	pages 365 – 370	(Applications: Edge detection)

Part 1: A quick practice run on how to perform 2D convolutions on image files

For Part 1, we will use these 2 matrices to explore 2D convolutions !

$$\text{Input photo} \qquad\qquad \text{2D filter}$$

$$X = \begin{matrix} \text{A photo containing} \\ \text{an array of black circles} \end{matrix} \qquad H = \frac{1}{2}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

1) Load this image using the matlab command:

```
X = imread('lattice_bigCircles_big_grid.tif');
```

2) Using matlab's *imshow* command, immediately plot your original image to make sure that you've loaded it correctly.

3) Using matlab's *double* command, convert your matrix X so that you can do math on it !

$$\underset{\left(\begin{matrix}\text{unsigned 8} - \text{bit integer} \\ \text{for raw photo}\end{matrix}\right)}{X} \xrightarrow{\text{convert}} \underset{(\text{double} - \text{precision floating point})}{X}$$

4) Now, let's check out the maximum / minimum pixel intensity values for your image by plotting a *pcolor* version of matrix X. Add a colorbar to your figure, and label your axes as:

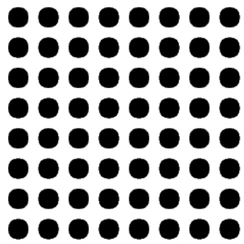
i) Horizontal axis label: "x-axis (pixels)"
ii) Vertical axis label: "y-axis (pixels)
iii) For this example, it's easier to see what going on by using the "copper" colormap:

```
colormap(copper)
```

** You will immediately see that your image seems to be "upside-down…." Let's use our favorite command to rectify this situation !! =)

```
set(gca, 'YDir', 'reverse');
```

5) We are now ready to perform 2D convolutions ! Using the *conv2* command, convolute image X with filter H and store the post-filtered result in a matrix called Y.



$$H[m,n] \;=\; \frac{1}{2}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

Output

$$Y = H \ast X$$

$Input: \quad X[m,n]$

$Filter\ impulse\ response$

6) Next, plot a *pcolor* version of your post-filtered image Y. Then:

a) Add a colorbar to your figure
b) For ease in visualizations, limit your colorbar axis color shadings to see the finer details:

```
caxis ( [ -250  250] )
```

c) Then, apply the "copper colormap" to your pcolor image.

** Important: Notice that your filtered pixel will vary from from *negative* values to positive values, and at the same time, your global average value is close to zero (most of the image is medium-brown). *The global zero-average value* comes from the fact that the sum of all entries within your H - matrix is zero !

$$H \;=\; \frac{1}{2}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix} \;\rightarrow\; \boxed{The\ sum\ of\ all\ 9\ entries \;=\; 0}$$

7) We're almost there ! Now, using matlab's *uint8* function, convert our filtered image Y into unsigned 8-bit integers (such that you can plot the image using *imshow*)

$$\underset{(double-precision\ floating\ point)}{Y} \;\xrightarrow{convert}\; \underset{\begin{pmatrix} unsigned\ 8-bit\ integer \\ for\ raw\ photos \end{pmatrix}}{Y}$$

Note: When you do this conversion, all negative numbers in matrix Y will now be suppressed to zero !

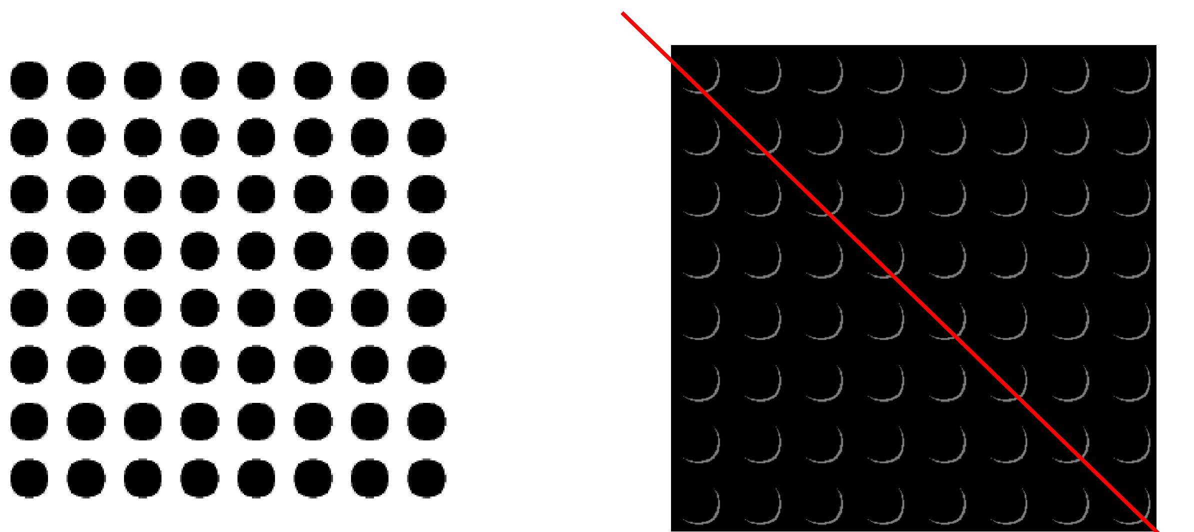
8) Finally, using matlab's *imshow* , plot your filtered image. Indicate that this photo is the post-filtered image by adding a title to this figure.

** <u>Note</u>: If you compare your post-filtered pcolor plot versus your final *imshow* image, you'll see that in regions where the pcolor values were negative , the *uint8* function has suppressed all those values down to zero.

… and this is why your final image appears mostly black !! =)

Part 2: The connection between our filter $\boldsymbol{H}$ $\longleftrightarrow$ $\boldsymbol{calculus}$!!!

Ok – we've filtered our input image, and you may have notice that H somehow accentuates the *<u>edges</u>* of our black circles………. and it only does so in a very particular direction !



$Original\ X[m,n]$ $Filtered\ \ Y[m,n[$

Uni-directional edge detection

<u>Figure 1</u>: Graphical representations between the original image and the filtered output

1) Given the above observation presented in Figure 1, give me a plausible, mathematical (calculus-based) explanation on why the action of $H[m,n]$ is to highlight uni-directional edges on our input data. You can type in your reasonings by using the *disp( )* function to echo your answers.

Hint #1: Stare at your matrix for a second:

$$H = \frac{1}{2}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix} \rightarrow \quad \textit{I think there is some kind of a directionality to it} \quad =)$$

Hint #2: You know that if you were gonna do flip & slide 2D convolutions on your input data matrix X, you would have to first mirror-image matrix H in both the horizontal + vertical directions before you do your "slides."

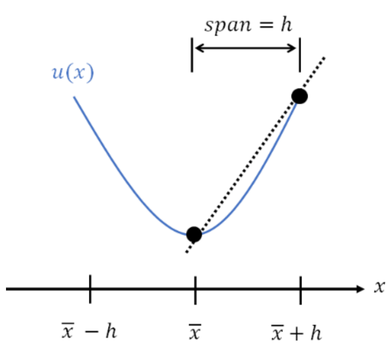
$$H[p,q] = \frac{1}{2}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix} \xrightarrow{mirror} \frac{1}{2}\begin{bmatrix} 0 & 0 & -1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \xrightarrow{mirror} H[-p,-q] = \frac{1}{2}\begin{bmatrix} -1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

You would now slide this guy across your input image X

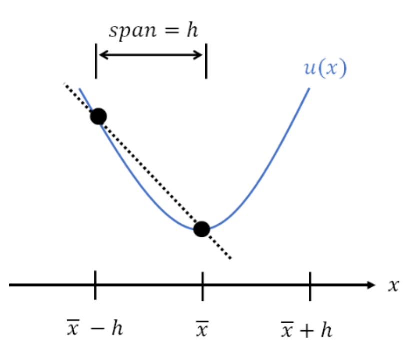
Hint #3: From class, we talked about the fact that we can use derivatives for edge-detection of signals, right ? Now, let's think: Stare at the 3 finite difference diagrams below… and ask yourself:

Which finite- difference approximations best matches what we have for the expression of $H[-p,-q]$??

-1 (center)	+1 (right)



-1 (left)	+1 (center)



-1 (left)	0 (center)	+1 (right)