Cleaning Up Image Processing

QEA

Spring 2016

Correlation in Facial Recognition

Kinds of Correlation in an Image Set

If we think about pictures now, we can think about two different correlations: the correlation between a given pair of pixels across all the pictures in a data set, and the correlation between pictures in a dataset across all the pixels in the images.

At your table, talk about what each of these correlations *means*. What would a high correlation between a given pair of pixels mean? What about a high correlation between a given pair of images?

Consider five pictures, each with a resolution of m x n pixels.

1. What is the size of the data matrix containing these five pictures as the columns?
2. What is the expression for the correlation matrix between the pictures? What size is this correlation matrix?
3. What is the expression for the correlation matrix between the pixels at different locations within the set of pictures? (Note that in order to compute an accurate correlation coefficient, you need to have multiple data points in each set being correlated. Also, pay careful attention to the mean and standard deviation you are using!). What is the size of this correlation matrix?
4. People's faces are approximately left-right symmetric. How would you expect this to affect the entries in the correlation matrix?

Test your understanding...

1. Pull in a few images from the classdata matrix and use the `resample` command to bring them down to a smaller resolution.
2. Now reshape them appropriately to create a matrix in which each column is a (reshaped) face.
3. Find the correlation between different images. Which images have the highest correlation? What is the correlation coefficient between your two images?

4. Now find the correlation between pixels across images (note that you'll want to use at least five different images for this to be at all meaningful) Can you explain what you see? Try taking a single column of this matrix and reshape that column into an image. What does that image tell you?

From Data to Dimensions

Thus far we've made a distinction between vectors as representing points in space, and vectors as representing data (e.g., a list of intensity values for pixels). But if you wanted to, there's no reason you couldn't (and it fact would be very powerful *to* think of a vector of n data points as representing a point in n dimensional space. If you did this, you could define all kinds of interesting things... For example, you could ask what the magnitude of the vector was; what the distance from one vector to another was, what the dot product of two vectors was, etc.

As a thought experiment, think about the following questions with your table mates.

1. Imagine you have images that are 100×100 pixels. If you represent each image as a vector, how large is the vector space?
2. If the intensity of pixel ij is a_{ij}, come up with an expression for the "magnitude" of the vector that describes the image A.
3. Come up with an expression that represents the "distance" between images A and B.
4. What does $A^T B$ tell you? What does this correspond to in vector operations?
5. Now that you've thought through this about, load three face images, and calculate the distance between each pair (i.e., take the difference between the two images, square each element, then sum up all the squared elements and take the square root of the sum). Do your answers make sense?

Smoothing and Downsampling

You should try out applying kernels to images. Here are the steps:

1. At your table, discuss how you would perform a kernel calculation in MATLAB for a 2D array. Discuss this plan at your table: how many loops might you need? How much could you reduce the number of loops if you used array operations and some of the built-in MATLAB commands (e.g,. `sum`)? How would you make it so you could use whatever size kernel you wanted?

2. Once you're pretty confident about how to go about it, write a short MATLAB code that calculates the resulting 2D array produced by operating on a grayscale image with a 2D kernel.

3. Test your code using some kernels that you feel confident about. For example, here's one you should feel VERY confident about:

$$\mathbf{K} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

4. Then use your code to smooth an image by doing averaging. Display the original image and the resulting image for each of the kernels. Was your intuition right?

5. Finally, try to create a new image by *downsampling*. Downsampling in images is a pretty simple idea: you're just selecting every nth pixel in the x direction and every mth pixel in the y direction.

6. Once you've manually tried downsampling, look at the documentation for `resample`, and try it out.

7. How far can you downsample a face and still recognize it? How does downsampling relate to kernel operations?

Image rotation and scaling

Let's put together some of the pieces we've already worked on. Say you have an *image* that you want to rotate by some angle θ. How would you do this?

1. Linked to the QEA module 2 website, you'll find a matlab workspace file called smiley_withpoints.mat. This file contains the smiley face example image you worked with a bit on the first day. This type of image is called a 'binary' image because it consists of essentially either 1's or 0's (in this case the 1's are 255's to play nice with standard image display). Open up the workspace and look at the smiley matrix. Now display it using the Matlab 'image' command.

2. Image files are matrixes of values corresponding to the pixel color: in this case 0's are black and 255's are white. The position information is encoded in the positions of these values within the matrix! In the case of a binary image like the smileyface, we can extract all of the image information as the set of matrix indices which have non-zero pixel values. Doing this in Matlab is very simple using the 'find' command. If you would like to play with this yourself, go ahead and 'help find' to read about the syntax and you can pull out the index information, but otherwise, I've done this for you, and it is in the workspace as the items 'XCoords', 'YCoords' and the combined 'Points' which is just the concatenation of 'XCoords' and 'YCoords'. Plot these points using the Matlab 'plot' command (you probably want to have it plot the points with symbols like 'x' or 'o' rather than the default line which will look really strange).

3. Alas! The extracted point information for matrix indices counts from the upper left-hand corner, whereas when we plot in the positive quadrant of a coordinate system, the origin is in the lower left-hand corner! So, our smiley face plots upside-down! Now, we could fix this by shifting the quadrant in which we plot the smiley face (plotting the y coordinates as negative) but it is much more fun to rotate the smiley face by 180 degrees. Can you do that? About which axis does it rotate? Can you shift the face to be centered on the origin and do rotations about the center of the face?

NOW, WHEN WE are working with a general grayscale image, we need to keep track of not just the location of pixels within the image, but also the intensity number associated with each position within the matrix. Let's work through rotating an image by 30 degrees. On the QEA website is a .mat file which has the transformer picture from your take-home exercises loaded and converted to a grayscale double image.

1. First sit down with a piece of paper and think about what you'd need to do, step-by-step to accomplish this rotation. Things to think about:
   - The origin of your image file will be at the lower left hand corner.
   - Your image has a certain size in pixels. Your transformed image will likely not occupy the same range of pixels.
   - You need to keep track of all positions within the image and all intensities associated with them, but in a format that you can act on with a transformation matrix.
2. Now, open up ShellforImTrans.m, a Matlab script I created to help you with this task. My solution for rearranging the image matrix (and putting it back together) is there. You may use it or write your own depending on your comfort level. Read through what I have done. Within the script is a place for you to put in your own transformation matrix definitions and operations. Try something simple first, like the identity matrix, to make sure you are comfortable using the script. Then start to work out the actual transformations you would use to rotate the image by 30 degrees. Please note that the final image in my script is restricted to the extent of the original image frame where noted. You can change this if you like, but it does put a limitation on how you do your transformation. Think about it!!

Other Image Processing

Yes, you've seen this in a previous day exercise, but you likely didn't get to it.. Please note that MATLAB by default uses matrix operations.

1. Consider the following matrix

$$\mathbf{R} = \begin{pmatrix} 0 & 0 & 0 & \cdots & \cdots & 1 \\ 1 & 0 & 0 & \cdots & \cdots & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 \\ \vdots & \cdots & \ddots & \ddots & \cdots & \vdots \\ 0 & \cdots & \cdots & \cdots & 1 & 0 \end{pmatrix}.$$

   (a) In words, describe what happens when you multiply another matrix by this matrix. To help visualize, you may wish to compute the following specific example

$$\begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

   (b) In MATLAB, load the giraffe image and after appropriate formatting, multiply the matrix representing the image by the matrix **R** defined above (with the appropriate dimensions). Repeatedly multiply the image with the matrix **R** 300 times. Display the resulting matrix as an image and explain what you see.

2. If you represent an image as a matrix, you can perform simple edge detection on the image by multiplying it with the following matrix.

$$\mathbf{E} = \begin{pmatrix} 1 & -1 & 0 & \cdots & \cdots & 0 \\ 0 & 1 & -1 & 0 & \cdots & 0 \\ \vdots & \cdots & \ddots & \ddots & \cdots & \vdots \\ -1 & \cdots & \cdots & \cdots & \cdots & 1 \end{pmatrix}$$

   Explain why multiplying a matrix representation of an image by the matrix **E** above detects edges. In MATLAB, load the giraffe image and after appropriate formatting, multiply the matrix representing the image by the matrix **E** above and display the resulting image (i.e., calculate and display **EG**). Is this what you expect?

3. This operation detects edges in one direction. Which direction does it detect in (horizontal or vertical)? Why?

4. Propose a way to detect edges in the other direction. Try your idea out.

5. Repeat the previous two parts with the image MilasHall.jpg. Describe the differences you see for the two different edge detections. Please explain any differences.

6. Now go back to your kernel code. Modify it to use a grayscale image of Milas Hall, and, apply the following kernel to the image data:

$$\mathbf{K} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

   Is the resulting image what you expect? Why?

7. Apply the following kernel to the giraffe image, and describe what you see.

$$\mathbf{K} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

   Can you explain what's happening here?

A first attempt at facial recognition

Now we are going to use some of the data tools we have learned here to take a first stab at matching up images. To start, load the class image data set and select two images from the set, and make sure you can do the following:

1. **Does each image have similar contrast and brightness?** If not, adjust them.
2. **Do the images** need to be rotated, or are they already aligned? If they need to be aligned, adjust them.
3. **Do the images** need to be shifted up or down, left or right? If so, adjust them.
4. **Analyze the two images**.
   (a) **How far apart** are your images in pixel space?
   (b) **How correlated are your two images?** Using what you have learned about correlation, you can compute the correlation coefficient between two images. Write a simple code which will take in two images and compute the correlation coefficient between them.

YOU NOW have the building blocks of a couple of rough facial recognition programs. Write a program that will take a test image as input, and then return the image from the class data set that matches it most closely. Can you write a program which will find either the distance in image space between your picture and each picture in the class database or the correlation coefficient between your picture and each other pictures in the class? Which picture is the best match? Is it one of your pictures?