Lab Session 4 Convolution and Clustering

Key aims of the Session:

- Better understand boundary conditions relating to convolution
- Gain a working understanding of k-means clustering
- Build a code file of and evaluate error measures

Contents

1: Boundary Conditions..... 1
2: Build you own K-Means clustering algorithm..... 3
3: Calculating Error..... 3

Editor vs Command Window

While you can complete this worksheet by entering commands sequentially into the command window, you may find it more useful to save and run the commands as a script. To open up the script editor, type `edit` into the command window. Don't forget to save your script so that you don't lose it. Once saved, you can run your script by clicking "Run" on the top ribbon or by typing the script's name into the command window. Please note that MATLAB script filenames should have a .m extension.

1: Boundary Conditions

In this section, we will look at boundary conditions, which can significantly affect the validity of our results.

The strict definition of convolution covers an infinite range and a lot of mathematics is done based on this definition:

$$(f * g)(\boldsymbol{x}) = \int_{-\infty}^{\infty} f(\boldsymbol{y}) g(\boldsymbol{x} - \boldsymbol{y}) \mathrm{d}\boldsymbol{y}$$

In other words, we should really start our sliding window at $(-\infty, -\infty)$ rather than the top-left corner. But we don't have information for this, so we have to estimate what is happening outside of the image.

To take an example, import the image "LancsLogo.jpg" (or another image of your choice) to Matlab, convert it to double, divide by 255 and take the mean across the red, green and blue colour channels, storing it as the variable `im`.

We will also need a kernel, e.g.

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

We created this last week with: `ker = [-1, -1, -1; -1, 8, -1; -1, -1, -1];`

Typically, boundary conditions are addressed by either:

1. Ignoring the boundary region. I.e. if your kernel size is $(2m+1) \times (2n+1)$ then ignore the first and last m rows, and the first and last n columns of the convolved matrix. Try this with the default matlab convolution to create a blurred image `cim`, i.e.:
   ```
   cim = conv2(im,ker,'same');
   ```
   Calculate the mean intensity of the trimmed matrix. What do you notice? Is the mean intensity of cim equal to the product of the mean intensity of the image and the mean intensity of the kernel?
2. Assuming that the intensity values outside of the image are equal to zero. You can try this with:
   ```
   tmp = 0*im;
   ime = [tmp,tmp,tmp;tmp,im,tmp;tmp,tmp,tmp];
   cime = conv2(ime,ker,'same');
   cim2 = cime(sz(1)+1:2*sz(1),sz(2)+1:2*sz(2));
   mean(cim2,'all')
   ```
   What does the second line achieve?
   How does this mean intensity compare with the mean intensity of cim?

Periodic Boundary Conditions

An alternative way to consider the space outside of the image is to consider the image as repeating. I.e. `ime = [im, im, im; im, im, im; im, im, im];`

Try displaying this as an image to see what it looks like.

Carry out the convolution `cime = conv2(ime,ker,'same');` and see what it looks like.

Extract the central image:

```
cim2 = cime(sz(1)+1:2*sz(1),sz(2)+1:2*sz(2));
```

Again, calculate the mean intensity and see how it compares with the mean intensity of cim. Is it closer to what you'd expect now?

Neumann Boundary Conditions

A second alternative is to consider the space outside the image as a reflection of the image. This can be particularly effective with small kernels.

Create the extended image:

```
ulim = rot90(im,2);
uim = flipud(im);
lim = fliplr(im);
ime = [ulim,uim,ulim;lim,im,lim;ulim,uim,ulim];
```

Display *ime* to see what it looks like.

Calculate the convolution and extract the central images

```
cime = conv2(ime,ker,'same');
cim2 = cime(sz(1)+1:2*sz(1),sz(2)+1:2*sz(2));
```

Again, calculate the mean intensity and see how it compares with the mean intensity of cim. Is it closer to what you'd expect now?

Try doing the same with some other images included in the folder. Consider which boundary conditions are most appropriate for each image.

2: Build you own K-Means clustering algorithm

For this task, we'll use the Iris dataset, so we'll need to load the `fisheriris` data file.

1. Since we know we have three classes, we'll use $k = 3$. Create 3 variables with randomly-chosen values to act as your initial cluster centres.
2. Use KNN (either with the inbuilt Matlab code or using your own from week 3) to assign each datapoint to a cluster.
3. Calculate the average values of the points in each cluster to determine new cluster centres.
4. Repeat steps 2 and 3 until there is no more change.
5. Visualise your results.

Try running again with different initialisation to see the change in results.

Consider how you can change this to use a for loop over steps 2 and 3.

3: Calculating Error

In lectures, we have considered many methods of calculating accuracy of error of our results, including accuracy, sensitivity, specificity, precision, negative predicted value, F1 score, etc.

Write your own code that can take classification results and ground truths as inputs and gives the error metrics as a vector of outputs. This should include the metrics mentioned above but you are free to add as many as you wish.

Try it out with your KNN code from Week 3.