Lab 2A: MATLAB Skills II

CM30080

Dr Matthew Brown

Introduction

This worksheet will guide you through some more advanced MATLAB usage needed for the subsequent recognition lab. When you have finished this worksheet you can move on to Lab 2B: Face Recognition.

Vectorising Image Data

In order to build classifiers, we often need to compute distances between images, for example the sum squared distance between two colour images $I(x, y, b)$ and $J(x, y, b)$ could be computed explicitly as:

$$\sum_x \sum_y \sum_b (I(x, y, b) - J(x, y, b))^2$$

This would require 3 for loops in MATLAB, i.e.,

```
sumsq = 0;
for i = 1:nRows
    for j = 1:nCols
        for b = 1:nBands
            diff = I(x, y, b) - J(x, y, b);
            sumsq = sumsq + diff*diff;
        end;
    end;
end;
```

This is very slow, since MATLAB is an interpreted language. It is much quicker to convert the image into a single vector and compute the distance over that, for example, the following code has exactly the same effect but is much faster:

```
sumsq = sum((I(:)-J(:)).^2);
```

The colon operator `(:)` used alone extracts all the data into a single 1D vector. Try this using random `100x100x3` images (`rand(100, 100, 3)`). What is the speedup? (you can find the time elapsed by enclosing commands in `tic` and `toc`, i.e.,

```
tic;
sumsq = sum((I(:)-J(:)).^2);
toc;
```

Avoiding For Loops

A similar trick is to try to avoid for loops where possibly by vectorising. One example of this is to use `repmat`, which can be used to repeat copies of a matrix or vector. Try the following code, which computes the distances between all columns of a matrix and stores them in matrix `D`.

```
nData = 400;
nDims = 1000;
data = rand(nDims, nData);

D = zeros(nData, nData);

% for loop
tic;
for i = 1:nData
    for j = 1:nData
        diff = data(:, i) - data(:, j);
        sumsqdiff = sum(diff.^2);
        D(i, j) = sumsqdiff;
    end;
end;
toc;

D2 = zeros(nData, nData);

% repmat
tic;
for i = 1:nData
    diff = data - repmat(data(:, i), 1, nData);
    sumsqdiff = sum(diff.^2);
    D2(i, :) = sumsqdiff;
end;
toc;

sum(sum(D - D2))
```

Convolution for Template Matching

When discussing template matching we often talk about *convolving* an image with a template, even though we actually don't reverse the template and multiply, as strictly speaking convolution does. This becomes a problem if we attempt to use `conv2` for template matching, because it will rotate the template by 180 degrees. Luckily MATLAB has a built-in function called `filter2` that corrects for this 180 rotation. Try the following example with

a test image `im` and smaller template image `template` (which could be a portion of the test image, for example), using `filter2` and `conv2`:

```
dotProd = zeros(imRows, imCols);
for i = 1:nBands
    dotProdi = filter2(template(:, :, i), im(:, :, i), 'same');
    dotProd = dotProd + dotProdi;
end;
```

Boundary Effects

Another subtlety with MATLAB convolution is what happens at the boundary of the image. In locations where the convolution template or kernel completely overlaps the image we simply sum the products of template and image, e.g.,

```
sum = 0;
for i = 1:tRows
    for j = 1:tCols
        sum = sum + template(i, j) * im(x+i, y+j);
    end;
end;
```

However, when the template extends outside the image, this will not work. One solution is simply to throw away these pixels, where the template extends outside the image, which can be done in MATLAB by using the `'valid'` argument to `conv2` or `filter2`, i.e.,

```
filter2(template(:, :, i), im(:, :, i), 'valid');
```

This however decreases the size of the image each time we convolve, so is not very satisfactory especially for large kernels. A good solution to this problem is to surround (*pad*) the image with plausible values before convolution, and then remove this area after. This can be accomplished using the MATLAB command `padarray`:

```
padSize = 10;
im = padarray(im, [padSize padSize], 'replicate');
```

which extends the image by replicating the edge pixels. Try convolving a small image with a large smoothing kernel, such as $\begin{bmatrix}1 & 6 & 15 & 20 & 15 & 6 & 1\end{bmatrix}/64$ and using `padarray` to prevent dark shadows appearing at the borders.

Note that you'll then have to re-select the central part of the image if you want to maintain the size, i.e.,

```
im2 = im((padSize+1):(padSize+nRows), (padSize+1):(padSize+nCols));
```

where `nRows` and `nCols` were the number of rows and columns before padding.

Conclusion

This worksheet has introduced more advanced MATLAB concepts needed for the subsequent recognition lab. You should now be ready to attempt Lab 2B: Face Recognition.