where $W$ are the averages of the pixel values in $I$—and therefore the correlation coefficient is a number between $-1$ and $1$, where $-1$ indicates perfect negative correlation, $1$ indicates perfect positive correlation, and $0$ indicates no correlation.

The terms in $\rho(r,c)$ can be collected into two matrices $A$ and $B$ of size $(m-p+1) \times (n-q+1)$, while $g$ is a single number. So the matrix $R$ that collects all the correlation coefficients $\rho(r,c)$ can be computed with the following MATLAB instruction once $A$, $B$, and $g$ are available:

$$R = A ./ (sqrt(B) * sqrt(g));$$

This is unsafe, because some entries of $B$ could be zero. To prevent division by zero, we can select the nonzero entries of the denominator and work only with those:

```matlab
 denom = sqrt(B) * sqrt(g);
 nz = denom ~= 0;
 R = zeros(size(A));
 R(nz) = A(nz) ./ denom(nz);
```
Figure 1 The sum of pixel values in the rectangle between rows \( r, s \) and columns \( c, d \) in the image \( I \) on the left can be written by combining values in the four shaded pixels in the integral image \( J \) on the right. Shaded pixels in \( J \) contain the integrals over the correspondingly shaded regions in \( I \) (shaded regions in \( I \) overlap, and start at pixel \((1, 1)\)).

In this exercise, we will find efficient ways to compute \( A, B, \) and \( g \). This is not the whole story: The computation could be made even more efficient by using Fourier transform ideas, but these are beyond the scope of this course.

If you try out snippets of the code you write on the images provided with this assignment, make sure you first cast all images to double.

1. The average \( m_T \) in equation (2) is a single number, while the image \( m_I \) has \( m - p + 1 \) rows and \( n - q + 1 \) columns. The image \( m_I \) can be written as the convolution of \( I \) with a kernel \( H \) of size \( p \times q \). What is the kernel?

2. The kernel \( H \) is separable. Show how it can be written as the product
   \[
   H = vh^T
   \]
of a column vector \( v \) and a row vector \( h^T \).

3. Ignoring image boundaries, approximately how many sums are required (in terms of \( m, n, p, q \)) to compute \( m_I \)? Do not count multiplications, as these are multiplications by one, and can be avoided. Also ignore any divisions.

4. Write a single MATLAB instruction that uses conv2 to compute the image \( m_I \) from \( I, p, \) and \( q \). This is just as an exercise, as we will find a more efficient way to compute this image later. Use separability, and make sure that the resulting image \( m_I \) has the proper size.

5. Even the separable version of convolution you just found is wasteful for this simple kernel: When sums for neighboring output pixels are computed, many of the sums are the same for both pixels, and repeating them wastes computation. To avoid this waste, we introduce the important notion of an integral image, which requires about \( 2mn \) sums to compute. Once we have the integral image, computing a pixel of \( m_I \) requires three sums (or differences). So the total cost is \( 5mn \) sums, regardless of the values of \( p \) and \( q \).

We define the integral image under the restriction that array subscripts must start with 1. This makes the definition a bit more cumbersome than it would be if we allowed indices to start at zero, but implementation in Matlab is more straightforward. Given an \( m \times n \) image \( I \), the integral image \( J \) of \( I \) is an image of size \((m + 1) \times (n + 1)\). The first row, \( J(1,:) \) and the first column, \( J(:,1) \), are all zeros. In the rest,

\[
J(i+1,j+1) = \sum_{u=1}^{i} \sum_{v=1}^{j} I(u,v).
\]

In words, entry \( J(i+1,j+1) \) contains the sum of all the pixels above and to the left of pixel \((i,j)\) inclusive.

The MATLAB image processing toolbox has a function integralImage that does this computation. If you do not have access to that code, a simplified version of it is provided with this assignment (this version assumes a nonempty input of double data type).

Consider now the entries of the original image \( I \) in a rectangle of pixels where the top left pixel is at \((r,c)\) and the bottom right pixel is at \((s,d)\). Referring to figure 1, the sum of the pixels in this rectangle can be computed in the following steps:

- Add all the pixel values above and to the left of \((s,d)\) in \( I \). This is simply \( J(s+1,d+1) \).
In summary, 
\[ \sum_{u=r}^{s} \sum_{v=c}^{d} I(u, v) = J(s+1, d+1) - J(s+1, c) - J(r, d+1) + J(r, c). \]

The right-hand side of this equation involves just three sums (once we have \( J \)), regardless of how large the rectangle is!

Write an efficient MATLAB function

\[
\text{function } S = \text{rectangleSums}(J, p, q)
\]

that takes as input the \((m+1) \times (n+1)\) integral image \( J \) for some \( m \times n \) image \( I \), as well as two integers \( p \) and \( q \) with \( 1 \leq p \leq m \) and \( 1 \leq q \leq n \) and computes an image \( S \) of size \((m-p+1) \times (n-q+1)\) whose entry \( i, j \) is the sum of pixel values in \( I \) (not \( J \)) in the \( p \times q \) rectangle with top-left pixel \((i,j)\) and bottom-right pixel \((i+p-1, j+q-1)\):

\[
S(i,j) = \sum_{u=i}^{i+p-1} \sum_{v=j}^{j+q-1} I(u,v).
\]

You get full credit for this question if your code has no explicit loops or convolutions. If you use loops, you get a quarter of the credit, and you will waste quite a bit of time waiting for your function to finish its job. If you use convolutions, you will get half the credit.

Hand in both your code and the two matrices output by the following code:

\[
I = \begin{bmatrix}
2 & 1 & 3; & 0 & 4 & 1; & 0 & 3 & 5; & 1 & 3 & 2
\end{bmatrix};
\]

\[
J = \text{integralImage}(I);
\]

\[
\text{S2} = \text{rectangleSums}(J, 3, 2);
\]

\[
\text{S1} = \text{rectangleSums}(J, 1, 1);
\]

6. Write MATLAB code that efficiently computes the \((m-p+1) \times (n-q+1)\) matrix \(mI\) and the number \(mT\) as defined in (2). Do not use loops or built-in MATLAB convolution operators. Computing \(mT\) is a simpler matter than computing \(mI\), as \(mT\) is a single number rather than a matrix.

7. Write MATLAB code that computes \(g\) as defined in equation (5), and without using loops. Recall that the MATLAB expression \(M \cdot T\) squares all entries in a matrix \(M\).

8. The expression for \(B\) given in equation (4) can be rewritten as follows:

\[
B(r,c) = \sum_{i=1}^{p} \sum_{j=1}^{q} I^2(r+i-1, c+j-1) + pq m_T^2(r,c) - 2m_T(r,c) \sum_{i=1}^{p} \sum_{j=1}^{q} I(r+i-1, c+j-1).
\]

The last double summation is \(pq\) times \(m_T\), so that

\[
B(r,c) = \sum_{i=1}^{p} \sum_{j=1}^{q} I^2(r+i-1, c+j-1) - pq m_T^2(r,c).
\]

The summations in the last expression can be handled efficiently with a new integral image and \text{rectangleSums}. Write MATLAB code that computes the matrix \(B\), assuming that code in the previous answers has already been run.

9. If we let

\[
D(i,j) = T(i,j) - m_T,
\]

then the expression (3) for matrix \(A\) can be rewritten as follows:

\[
A(r,c) = \sum_{i=1}^{p} \sum_{j=1}^{q} [I(r+i-1, c+j-1) - m_T(r,c)] D(i,j)
\]

\[
= \sum_{i=1}^{p} \sum_{j=1}^{q} I(r+i-1, c+j-1) D(i,j) - m_T(r,c) \sum_{i=1}^{p} \sum_{j=1}^{q} D(i,j) - K(r,c).
\]

COMPSCI 527 — Duke — October 1, 2012
The expression for $K(r,c)$ above differs slightly from a convolution. How?

10. Write efficient MATLAB code that uses the Matlab function conv2 to compute the matrix $A$ from the expression given in the previous question. Assume that code you wrote in previous answers has already been run. Explain briefly how you use conv2, and do not use explicit loops. [Note: This step could be made even more efficient by using Fourier transforms. Ignore this fact.]

11. Put everything together and write a function that begins as follows:

```matlab
function R = correlation(I, T)
% Check that input arrays correspond to gray-level images
if ~ismatrix(I)
    error('Image must be gray')
end
if ~ismatrix(T)
    error('Template must be gray')
end
%
% Convert to double to do arithmetic
I = double(I);
T = double(T);

%%% Your code here %%%

Your function computes the $(m - p + 1) \times (n - q + 1)$ matrix $R$ with the correlation coefficients defined in equation (1) from the input $m \times n$ image $I$ and input $p \times q$ template $T$. The preamble above checks that images are gray (as opposed to color), and casts the inputs to type double so that all arithmetic operations can be performed. Remember to prevent divisions by zero.

Intersperse brief comments and hand in the entire code of your function.

12. Run the following code:

```matlab
T = [2 0; 3 1];
I = [2 0 0; 3 1 0; 1 -4 0; 1 -6 -2];
R = correlation(I, T)
```

This should result in the following matrix:

$$
R = \begin{bmatrix}
1.0000 & 0.7746 \\
0.3026 & -0.6404 \\
0.7980 & -1.0000 \\
\end{bmatrix}.
$$

Report what matrix $R$ you obtain from running this code. If your matrix differs by more 0.005 in any of its entries from the result above, you probably have a bug somewhere. You may want to iterate until you get the right result.

13. Report the time it takes to run correlation on the image I.jpg and template T.jpg as measured with the following code:

```matlab
I = imread('I.jpg');
T = imread('T.jpg');
tic; R = correlation(I, T); toc
```

Also state what CPU and operating system you ran your code on.

14. Look at $I$, $T$, and $R$ in MATLAB. The function imshow will work for $I$ and $T$ (before you convert them to double). To view an image with pixels of type double, such as $R$, do the following:

```matlab
figure
clf
imagesc(R)
colormap gray
axis image
axis off
```

Where is the global maximum of $R$, and what is its value? To find out, do the following:
Does the answer you obtain make sense to you?

15. Notes: Your ability to answer this question and the following ones depends on the code you developed so far being correct. In case this is not so, the correct correlation matrix $R$ is provided in file $R.mat$ with this homework assignment. You may want to use this function to check what you get from your code anyway. Small discrepancies (on the third digit or higher) are probably OK.

The Matlab image processing toolbox has a function normxcorr2 that computes the correlation coefficients. It would not be profitable for you to learn from that function how to solve the problems in this assignment. Should you want to compare your output with MATLAB’s, here is how to resize the output from normxcorr2 to match that from correlation:

```matlab
[p, q] = size(T);
C = normxcorr2(T, I);
C = C(p:(end-p+1), q:(end-q+1));
```

Why is there a maximum only on one package of M&M candy? Specifically, there are two more such packages in the image. Why are the correlation values for the other two so low? Explain separately for each of the two other packages.

16. Explain briefly how you could in principle obtain better correlation for all three packages of M&Ms. No need to implement anything, but please be both clear and brief.

17. There are secondary peaks around the global peak in $R$. Why?