Started on	Monday, 5 October 2020, 3:09 PM
State	Finished
Completed on	Tuesday, 6 October 2020, 12:49 PM
Time taken	21 hours 40 mins
Grade	**4.00** out of 4.00 (**100**%)
Feedback	**Well done!** **You have successfully passed this quiz.**

Information

Introduction

This coursework will continue to introduce you to working with images in MATLAB. In addition, it will allow you to explore some topics that are covered in this week's lecture.

Remember that commands which you can type at the MATLAB prompt are indicated by text in the following typeface:

```
matlab_function(parameter1, parameter2);
```

As in the previous coursework, you will work with the rooster and elephant images, plus the woods image. Copy the rooster.jpg, elephant.png and woods.png files from the module's KEATS webpage to the directory in which you are running MATLAB, and load these images into MATLAB by using the following commands:

```
Ia=imread('rooster.jpg');
```

```
Ib=imread('elephant.png');
```

```
Ic=imread('woods.png');
```

Information

Image Modification

In the previous coursework you learnt how to read values from an image. We can use similar methods to assign values to locations in an image.

For example, the previous coursework explained that to see the value at row 403 and column 404 of the elephant image, you would use:

```
Ib(403,404)
```

To change the value at row 403 and column 404 of the elephant image, you would use:

```
Ib(403,404)=1;
```

This will replace the previous value with a new value of 1.

We can select larger parts of an image using MATLAB's standard "colon" operator. We can also replace parts of an image using this technique. For example, the following will make a rectangular patch of the elephant image white:

```
Ib(401:end,401:end)=255;
```

Check this by displaying the image:

```
figure(1), clf
imagesc(Ib); colormap('gray')
```

We can go further and use this technique to create entirely synthetic images, e.g.:

```
Isyn=zeros(201,201);
Isyn(51:150,51:150)=1;
Isyn(81:120,:)=0.5;
Isyn(:,81:120)=0.75;
figure(2), clf, imagesc(Isyn);
```

Question 1 Correct Mark 0.80 out of 0.80

Image Resizing and Reshaping

Resizing, or scaling, an image can be done with the command `imresize`.

For example, to make copies of the (now modified) elephant image that are half and twice the original size, do the following:

```
Ibsmall=imresize(Ib,0.5);
Iblarge=imresize(Ib,2);
```

To see the results:

```
figure(3), clf
subplot(2,2,1), imagesc(Ibsmall)
subplot(2,2,2), imagesc(Iblarge)
```

Notice the difference in the axes labels.

When resizing an image we can choose which method MATLAB employs by using an optional parameter (some of these methods have been discussed in the lecture when describing demosaicing). For example, to make an image twice the original size using bilinear interpolation, you would use the following:

```
Iblarge=imresize(Ib,2,'bilinear');
```

To make an image half the original size using nearest-neighbor interpolation, you would use the following:

```
Ibsmall=imresize(Ib,0.5,'nearest');
```

Sometimes it is convenient to represent an image as a vector rather than as a matrix. If `I` is an image in matrix form, `Iv = I(:);` will create a vector `Iv` whose elements are the columns of `I` appended end-to-end to form one column vector. To return a vector to matrix form, use `reshape`. For example,

```
Isynv=Isyn(:);
Isynv(1:202:end)=1;
Isyn=reshape(Isynv,201,201);
figure(4), clf, imagesc(Isyn);
```

EXERCISE:

Report the pixel intensity values for the following imges and locations.

Pixel intesity at row=237, column=192 of `Ibsmall`: 101 ✓

Pixel intesity at row=237, column=192 of `Ib`: 25 ✓

Pixel intesity at row=237, column=192 of `Iblarge`: 144 ✓

Pixel intesity at row=474, column=384 of `Ib`: 101 ✓

Pixel intesity at row=948, column=768 of `Iblarge`: 101 ✓

Note that you should have found that the pixel values at the same coordinates in the three different sized images are different, while the values at the equivalent locations in the three different sized images are similar.
If you got the wrong answers, here is what you should have done:

Pixel intesity at row=237, column=192 of `Ibsmall`: `Ibsmall(237,192)`

Pixel intesity at row=237, column=192 of `Ibsmall`: `Ib(237,192)`

Pixel intesity at row=237, column=192 of `Iblarge`: `Iblarge(237,192)`

Pixel intesity at row=474, column=384 of `Ibsmall`: `Ib(474,384)`

Pixel intesity at row=948, column=768 of `Iblarge`: `Iblarge(948,768)`

Information

Image Output

In the previous coursework you learnt how to read an image file into MATLAB. You can also save a MATLAB image as a file. This can be achieved using the `imwrite` command. This command takes three arguments. The first specifies the name of the MATLAB variable that contains the image you wish to store, the second contains the name of the file you wish to store the image in, and the third specifies the file format you wish to use. For example:

```
imwrite(Ibsmall,'elephant_small.jpg','jpg')
```

will save the small version of the modified elephant image as a JPEG file.

Alternatively, you might want to save a MATLAB figure window (with the axes labels, figure title, etc.) as an image. This can be done using the MATLAB `print` command. You execute this command having selected the figure you wish to save, either by using the mouse, or by using the `figure` command. For example:

```
figure(3)
print -dpdf elephants.pdf
```

will produce a pdf file with the two subplots of different sized elephant images you produced earlier. By typing `help print` you can find out how to produce figures in other formats.

Alternatively, you might want to save the variables in your current MATLAB session so that you can continue work with them a later date. This can be achieved using the `save` command. For example:

```
save('cv_cw2_variables.mat');
```

will produce a file that you can later load using the `load` command to recover all the variables that you currently have in memory.

Question 2 Correct Mark 0.80 out of 0.80

Intensity Change Detection using Image Shifts

We will start off by converting the elephant image (modified to have a white rectangular region) from uint8 format to double format:

```
Ibd=im2double(Ib);
```

This is done because the following calculations will produce negative numbers that can not be represented using unsigned integer values, but can be correctly represented using floating-point numbers.

We then calculate the differences in intensity values between pixels that are vertical neighbours. To do this we take all the rows of the image that have a neighbour vertically below them, subtract that neighbour's value, and show the result:

```
Ibdiffv=Ibd(1:end-1,:)-Ibd(2:end,:);
```

```
figure(3), clf, imagesc(Ibdiffv);colormap('gray');colorbar
```

We calculate the differences in intensity values between pixels that are horizontal neighbours, using a analogous method:

```
Ibdiffh=Ibd(:,1:end-1)-Ibd(:,2:end);
```

```
figure(4), clf, imagesc(Ibdiffh);colormap('gray');colorbar
```

The results are two images in which most pixels have low intensity values (values near zero). The few high intensity values (positive or negative) that are significantly different from zero occur where neighbouring pixels have different intensity values. We informally refer to these locations as "edges", as they often occur at the boundaries or edges of different objects or surfaces.
Highlighting locations where intensity changes is often performed in computer vision, however, it is typically be achieved using alternative techniques to the one used here. These typical techniques will be desctibed in next week's lecture. Specifically, identical results to those produced here can be obtained using horizontal and vertical first-derivative masks, which will be described in next week's lecture.

We can combine the vertical and horizontal difference images into a single image showing (using positive pixel values) large changes in the intensity in both the vertical and horizontal directions, by calculating the L2-norm (or Euclidean-norm):

```
Ibdiff=sqrt(Ibdiffh(1:end-1,:).^2+Ibdiffv(:,1:end-1).^2);
```

```
figure(5), clf, imagesc(Ibdiff); colormap('gray'); colorbar
```

We can convert this to a binary image (using one of the conversion methods mentioned in last week's coursework), and display it, as follows:

```
bw=im2bw(Ibdiff,0.075);
```

```
figure(6), clf, imagesc(bw); colormap('gray'); colorbar
```

EXERCISE:

Report the pixel intensity values for the following images and locations.

Pixel intesity at row=505, column=400 of `Ibdiffv`: 0 ✓

Pixel intesity at row=505, column=400 of `Ibdiffh`: -0.6039 ✓

Pixel intesity at row=505, column=400 of `Ibdiff`: 0.6039 ✓

Pixel intesity at row=505, column=400 of `bw`: 1 ✓

If you correctly followed the instructions in this section then you would have produced the following images.

Figure 3

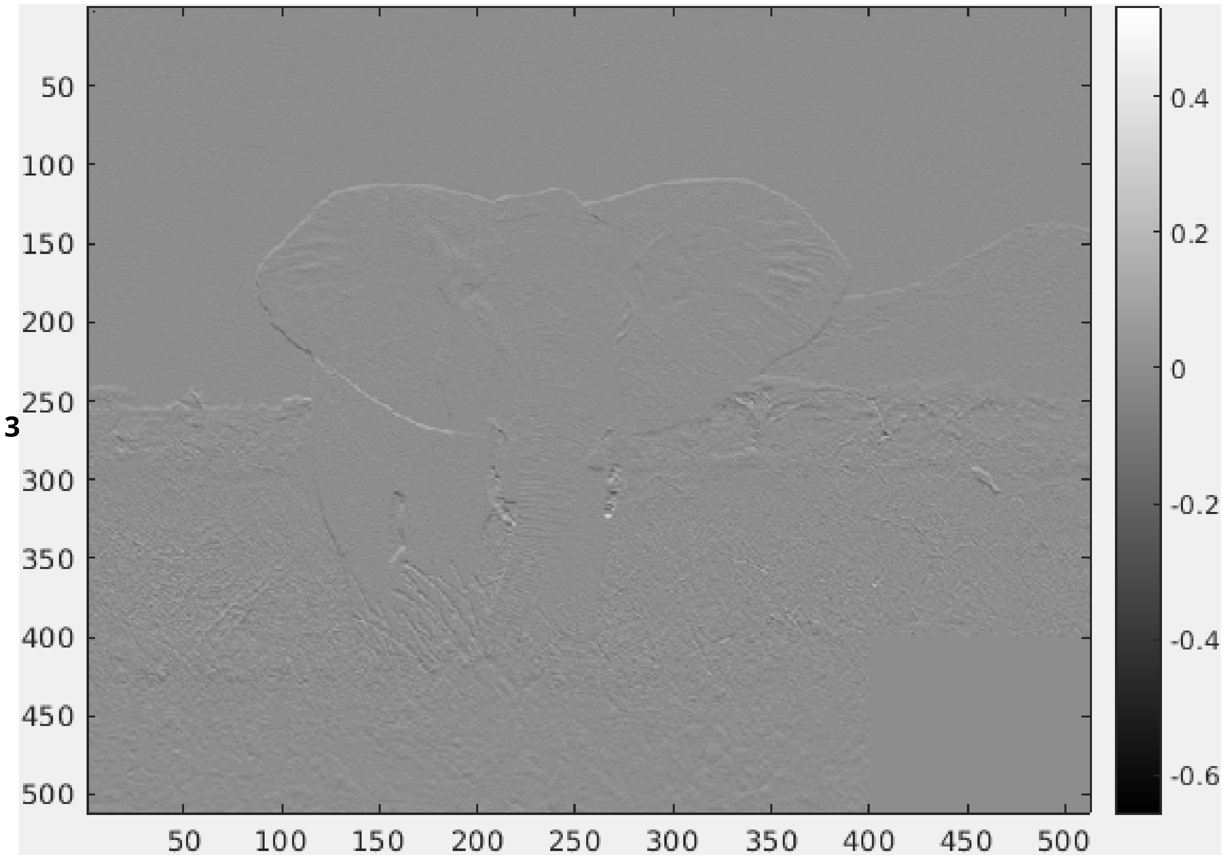
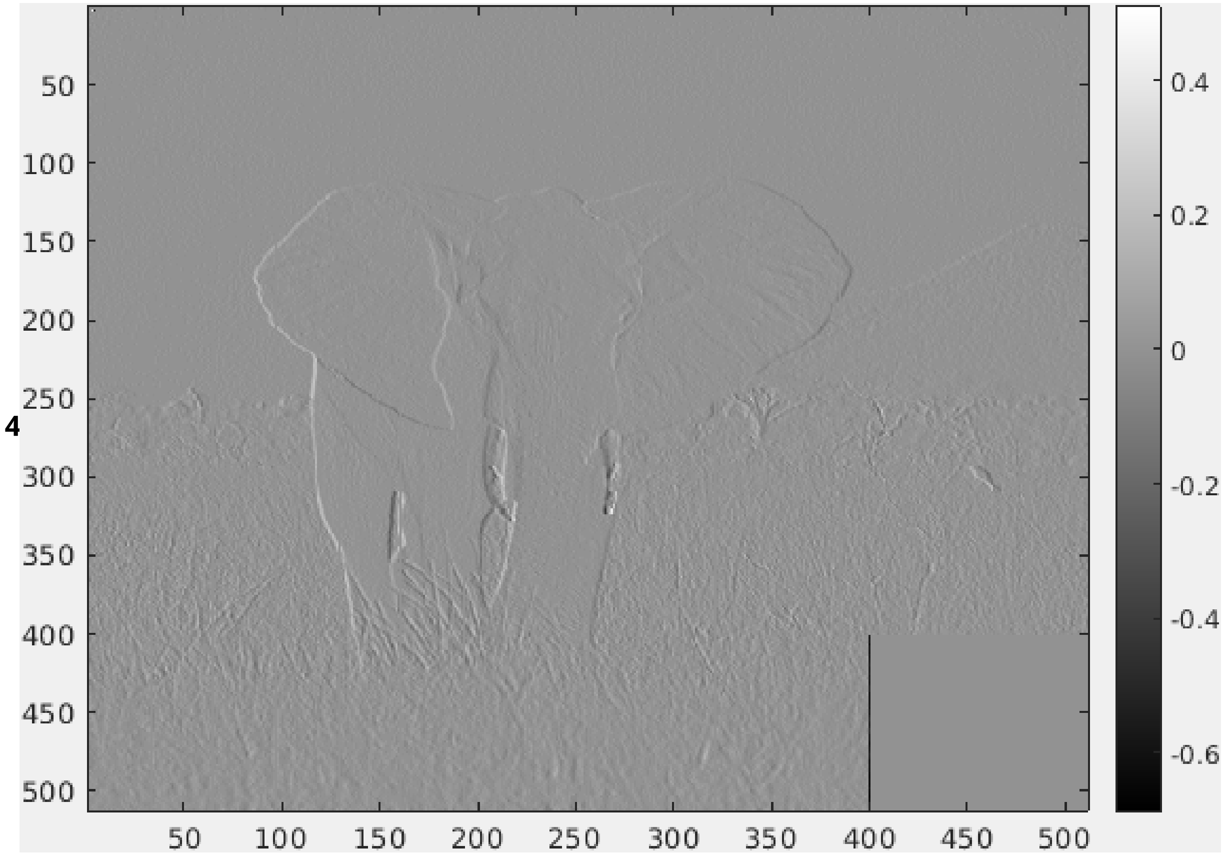
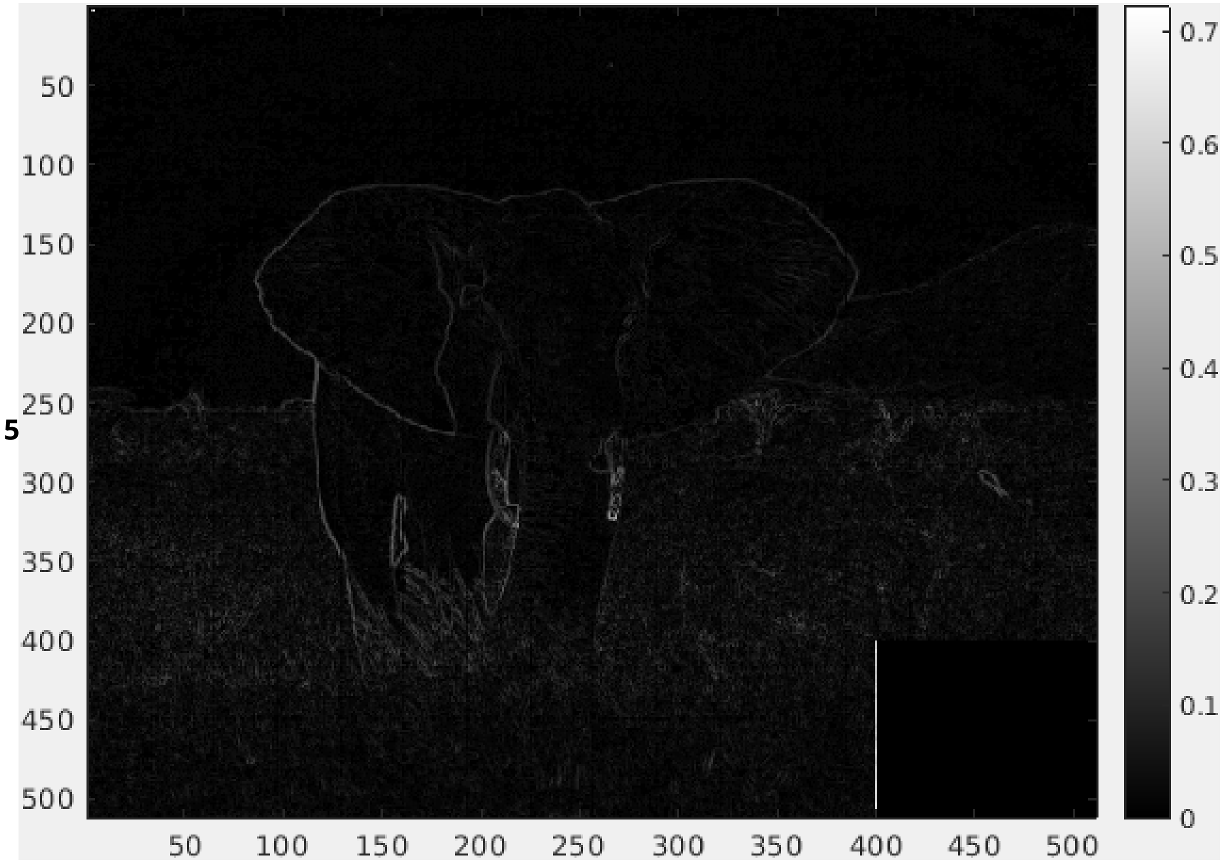


Figure 4



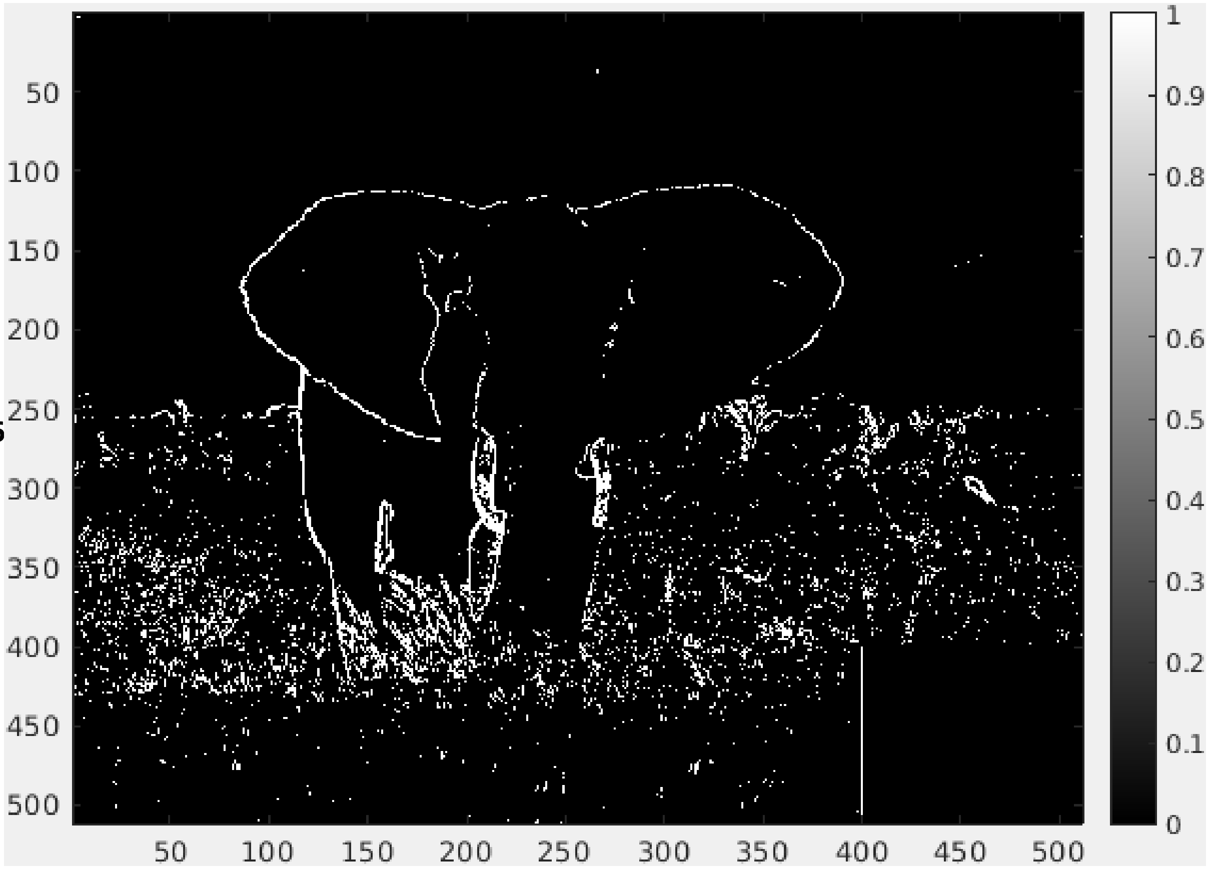
50
100
150
200
250
300
350
400
450
500
50
100
150
200
250
300
350
400
450
500
0.4
0.2
0
-0.2
-0.4
-0.6

Figure 5



50
100
150
200
250
300
350
400
450
500
50
100
150
200
250
300
350
400
450
500
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0

Figure 6



To get the requested pixel values you should have done the following:

Pixel intesity at row=505, column=400 of `Ibdiffv`: `Ibdiffv(505,400)`

Pixel intesity at row=505, column=400 of `Ibdiffh`: `Ibdiffh(505,400)`

Pixel intesity at row=505, column=400 of `Ibdiff`: `Ibdiff(505,400)`

Pixel intesity at row=505, column=400 of `bw`: `bw(505,400)`

Question 3 Correct Mark 0.80 out of 0.80

Redundancy in Natural Images

Natural images exhibit a high degree of redundancy, i.e. the similarity between the intensity of a pixel at location (x, y) and that at a neighbouring location (x + o, y) is inversly proportional to the distance (o) between these points.
One metric for assessing similarity is the correlation coefficient, which can be calculated for two equally-sized images (or image patches) using the MATLAB command `corr2`. Execute `help corr2` to find out how the correlation coefficent is calculated.
To calculate the correlation coefficient for the elephant image (modified to have a white rectangular region) with itself, we can use:

```
corr2(Ib,Ib)
```

In the previous section of this coursework we subtracted an image from a copy of the same image shifted one pixel vertically. Using this technique, calculate the similarity (measured using the correlation coefficient) for the modified elephant image and the same image shifted one pixel vertically. Write a simple MATLAB function or script that will automate the process of calculating the correlation coefficient between overlapping parts of the same image shifted verticlly by different values. Ensure that the two image parts are as large as possible. Using this code calculate the correlation coefficient for values of shifts between 0 and 30 pixels for both the elephant image (modified to have a white rectangular region) and the woods image. Plot a graph of correlation coefficient vs. shift for both the rooster and the woods image.

EXERCISE:

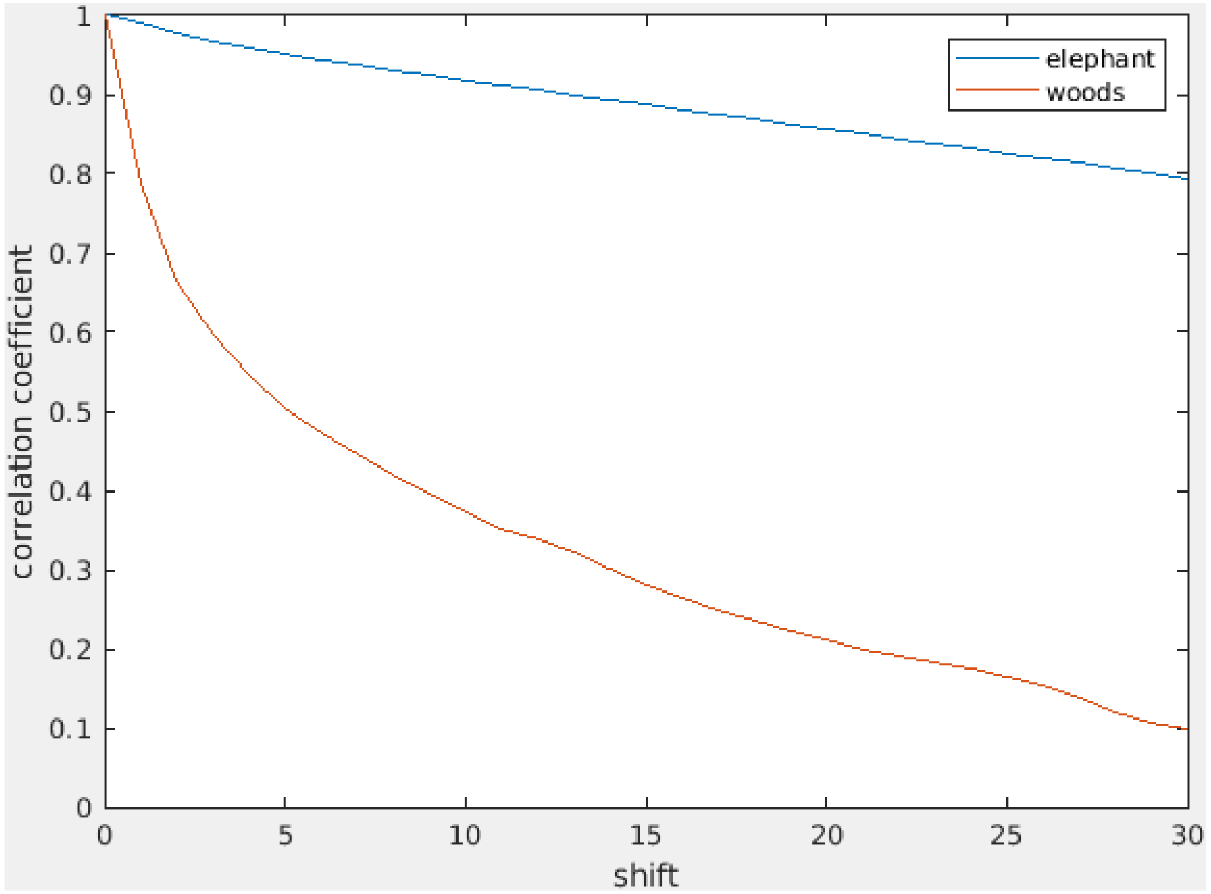
Briefly explain the results you have obtained.

Report the correlation coefficient you measured (correct to 2 decimal places) for the following shifts:
3 pixel shift for the elephant image 0.97 ✓ and for the woods image 0.60 ✓
18 pixel shift for the elephant image 0.87 ✓ and for the woods image 0.23 ✓

You should have written code equivalent to the following:

```
offsets=[0:30];
for offset=offsets
    simb(offset+1)=corr2(Ib(1:end-offset,:),Ib(1+offset:end,:));
    simc(offset+1)=corr2(Ic(1:end-offset,:),Ic(1+offset:end,:));
end
figure(7), clf,
plot(offsets,simb);
hold on;
plot(offsets,simc);
legend({'elephant','woods'});
xlabel('shift'); ylabel('correlation coefficient')
```

And produced a graph like this:



Observe that for both images the correlation coefficient falls as the shift increases. This is because nearby pixels tend to have similar intensity values (so high correlation) whereas pixels further away have less similar intensity values (so lower correlation). The correlation coefficient falls more slowly for the elephant image. This is because the elephant image contains larger regions of similar intensity values (like the sky and the elephant's ears).

You should have obtained the numerical values, as follows:
3 pixel shift for the elephant image=simb(4), and for the woods image=simc(4)
18 pixel shift for the elephant image=simb(19), and for the woods image=simc(19)

Question 4 Correct Mark 0.80 out of 0.80

Redundancy Reduction by Retinal Ganglion Cells (simulated with Difference of Gaussians)

Retinal ganglion cells reduce redundancy. To simulate a retinal ganglion cell we use a Difference of Gaussians (DoG) operator (or mask). To produce a DoG mask we can use the MATLAB command `fspecial` to create 2 Gaussian masks that are subtracted from each other. For example, to generate an on-centre, off-surround DoG:.

```
dog=fspecial('gaussian',9,1)-fspecial('gaussian',9,1.5);
```

To simulate the response of retinal ganglion cells to all different parts of an image we can convolve the DoG mask with the image (the details of convolution will be explained in the next week's lecture, but you can complete the current coursework without knowing these details). For example, to simulate the responses of on-centre, off-surround retinal ganglion cells to all different parts of the modified elephant image, execute the following:

```
Ibdog=conv2(Ibd,dog,'same');
```

Note, use `Ibd`: the version of the elephant image that was converted to double format earlier in this coursework.

Take a look at this result, by showing the image created:

```
figure(8), clf, imagesc(Ibdog); colormap('gray'); colorbar
```

Now repeat the process described in the preceding section to plot a graph of the correlation coefficients calculated between `Ibdog` and the same image shifted by 0 to 30 pixels vertically.
Repeat the above for the woods image after convolution with the DoG mask.

EXERCISE:

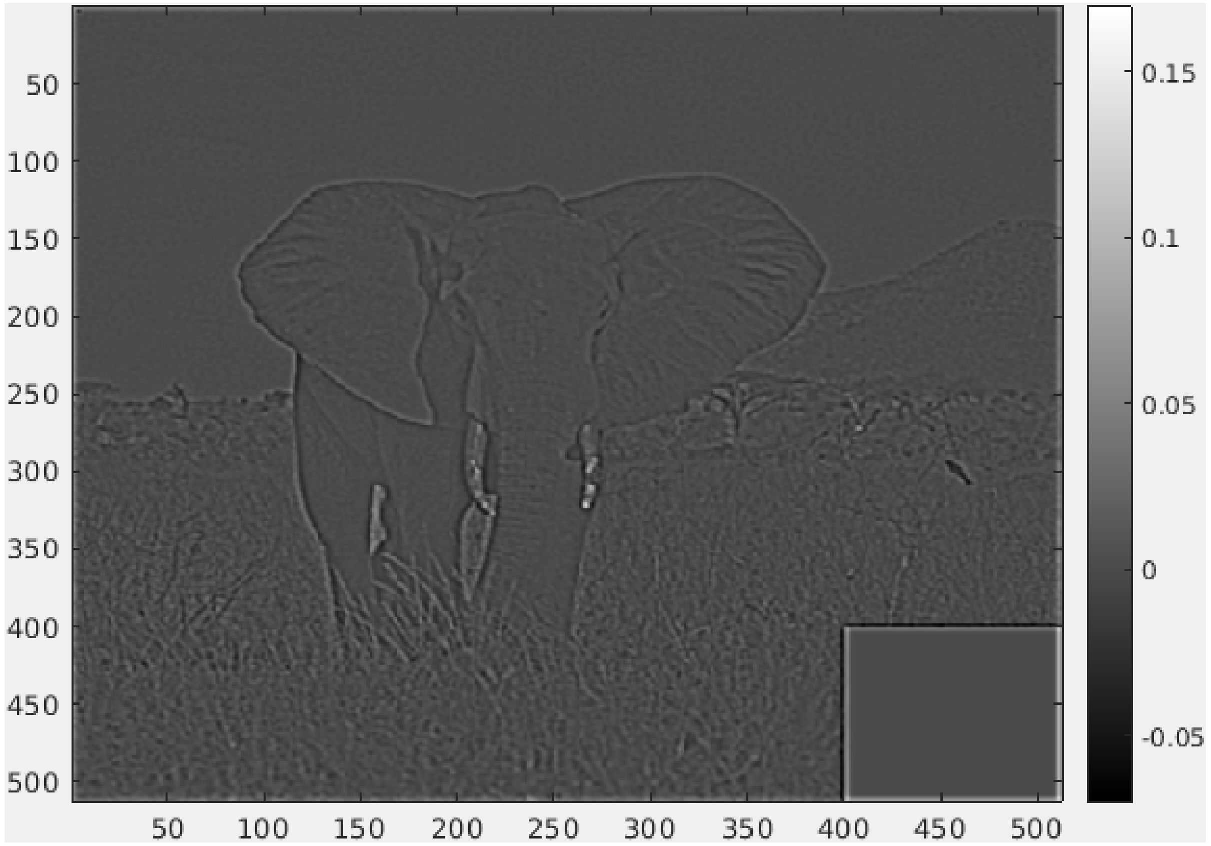
Briefly explain why the current results differ from those in the previous section.

Report the correlation coefficient you measured (correct to 2 decimal places) for the following shifts:

7 pixel shift for the elephant image 0.34 ✓ and for the woods image 0.14 ✓

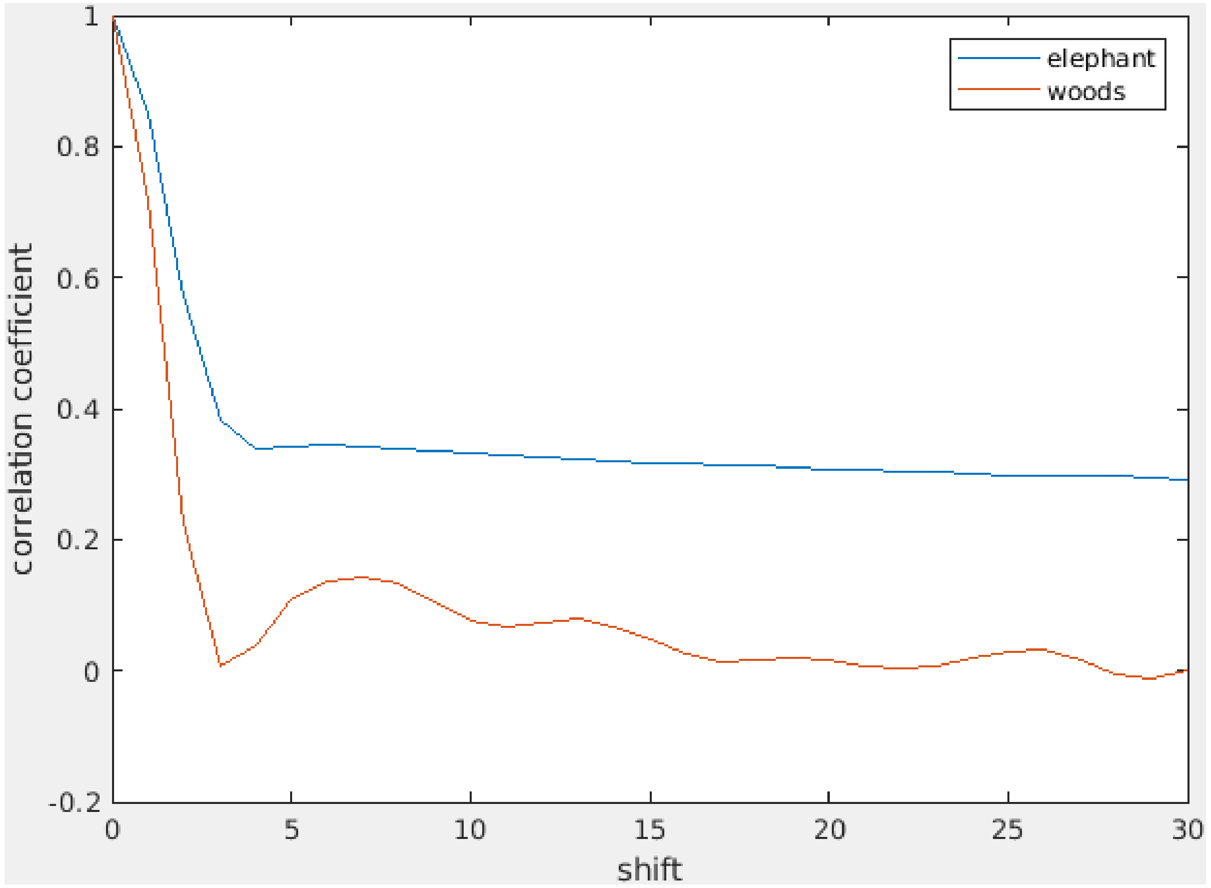
21 pixel shift for the elephant image 0.31 ✓ and for the woods image 0.01 ✓

You should have produced an image like this:



Observe that the retinal ganglion cells produce responses that resemble those found earlier in the section "Intensity Change Detection using Image Shifts". Specifically, many pixels have low intensity values (values near zero), and the highest intensity values (positive or negative) occur where there was a change in intensity value between nearby pixels, which often occurs at the "edges" of objects or surfaces. Unlike previously, both horizontal and vertical changes in intensity are shown in one image, rather than in two separate images.

You should have produced a graph like this:



Observe that the correlation falls with increasing shift, and this fall in correlation is quicker for the DoG filtered images, than for the original images. Many pixels in the DoG filtered image have values close to the average pixel vale, and hence, these pixels produce a correlation coefficient near to zero. Furthermore, pixel values that differ from the average pixel value often have neighbours that have very different values: i.e. their values are not correlated. In other words, the output of the retinal ganglion cells is less redundant than the input.

You should have obtained the numerical values, as follows:
7 pixel shift for the elephant image=simb(8), and for the woods image=simc(8)
21 pixel shift for the elephant image=simb(22), and for the woods image=simc(22)
where simb and simc are produced by code the same as the used in the previous section, but applied to the results produced by convolving the images with the DoG mask.

Question 5 Correct Mark 0.80 out of 0.80

Colour Detection with Colour Opponent Cells

The type of retinal ganglion cell simulated in the previous section processes intensity. In this section, a different type of retinal ganglion cell, colour opponent cells, will be simulated.

In the previous section, we created the DoG (by subtracting one Gaussian from another) and then convolved this with the image. It would have been possible to produce an identical result by convolving the image twice with two different Gaussian masks, and taking the difference between the two ouputs of these convolutions. This latter approach is used to simulate colour opponent cells, except each Gaussian is convolved with a different colour channel of a colour image.

We will apply this technique to the rooster image, which is an RGB colour image, after conversion from uint8 to double format, i.e using `Iad=im2double(Ia);`.

First create two Gaussians to simulate the centre and surround of the cell's RF:

```
gc=fspecial('gaussian',9,1);
```

```
gs=fspecial('gaussian',9,1.5);
```

You can then simulate the response of a red-on, green-off colour opponent cell by executing:

```
IaRG=conv2(Iad(:,:,1),gc,'same')-conv2(Iad(:,:,2),gs,'same');
```

Note colour channels are the third-dimension of `Iad` and channel 1 is the red channel, and channel 2 is the green channel.

Generate an image with 4 subplots, with the subplots showing the response of the following centre-surround colour opponent cell combinations on the rooster image:

1. red-on, green-off
2. green-on, red-off
3. blue-on, yellow-off
4. yellow-on, blue-off

Note: an RGB image does not have a yellow channel, but you can create one by calculating the mean of the red and green channels, using: `mean(Ia(:,:,1:2),3)`

EXERCISE:

Briefly explain the results you have obtained.

For each of the 4 images, determine the pixel intensity values at row=341, column=374 (correct to 2 decimal places)

red-on, green-off:	0.08	✓
green-on, red-off:	0.05	✓
blue-on, yellow-off:	-0.12	✓
yellow-on, blue-off:	0.23	✓

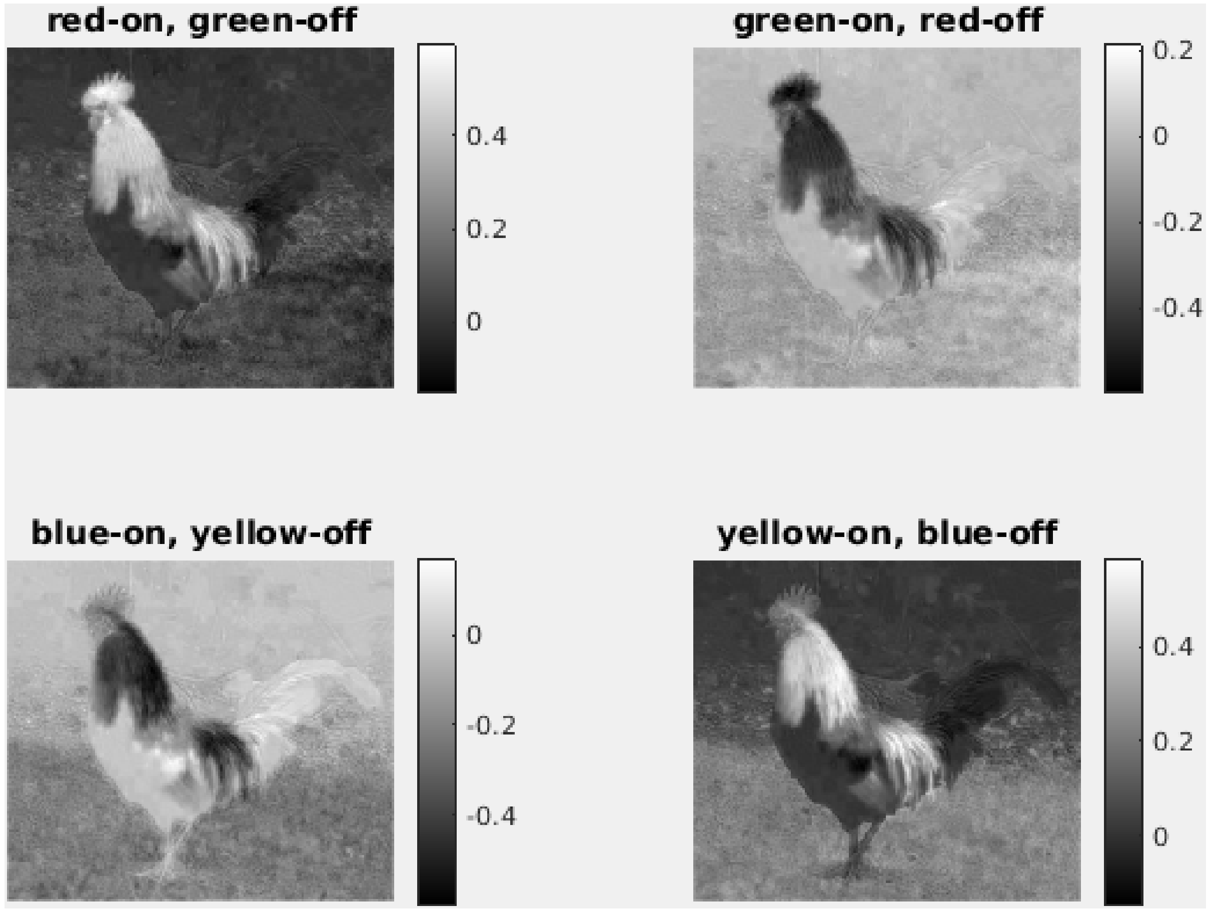
You should have written code equivalent to the following:

```
Iad=im2double(Ia);
gc=fspecial('gaussian',9,1);
gs=fspecial('gaussian',9,1.5);
figured(10), clf, colormap('gray')
IaRG=conv2(Iad(:,:,1),gc,'same')-conv2(Iad(:,:,2),gs,'same');
subplot(2,2,1); imagesc(IaRG); axis('off','equal','tight'); colorbar, title('red-on, green-off');
IaGR=conv2(Iad(:,:,2),gc,'same')-conv2(Iad(:,:,1),gs,'same');
subplot(2,2,2); imagesc(IaGR); axis('off','equal','tight'); colorbar, title('green-on, red-off');
Y=mean(Iad(:,:,1:2),3);
IaBY=conv2(Iad(:,:,3),gc,'same')-conv2(Y,gs,'same');
subplot(2,2,3); imagesc(IaBY); axis('off','equal','tight'); colorbar, title('blue-on, yellow-off');
```

```
IaYB=conv2(Y,gc,'same')-conv2(Iad(:,:,3),gs,'same');
```

```
subplot(2,2,4); imagesc(IaYB); axis('off','equal','tight'); colorbar, title('yellow-on, blue-off')
```

You should have produced an image like this:



Observe that a X-on, Y-off cell produces high positive output where the colour of the image matches the colour of the on-channel (i.e. X). For example, a red-on green-off cell produces high output at locations in the image where the colour is red (e.g. on the neck and head). The cells find differences in intensity between different colour channels, this allows them to respond when the intensity in a specific channel is high (which corresponds to the presence of a particular colour) rather than responding when there is high intensity across multiple channels (as would occur when the image is white).