Digital Media Processing
Tutorial 4:

Binary Images

Dr Samuel Smith
samuel.smith@bcu.ac.uk

BIRMINGHAM CITY
University

Over the last tutorials we have covered reading images, displaying images, converting images, addressing individual image pixels and even histogram calculations. This has allowed us to become more familiar with both the syntax on Matlab and the ease of accessing and adjusting simple image data. In this tutorial we will start to look into the theory and application of a key process within image processing, namely in Morphology and the specific operations of "Erosion" and "Dilation".

Prerequisites: Tutorial 1,2,3,4 . It is essential that you have completed all previous tutorials and have completed all the independent tasks for each tutorial. These need verifying by the tutors.

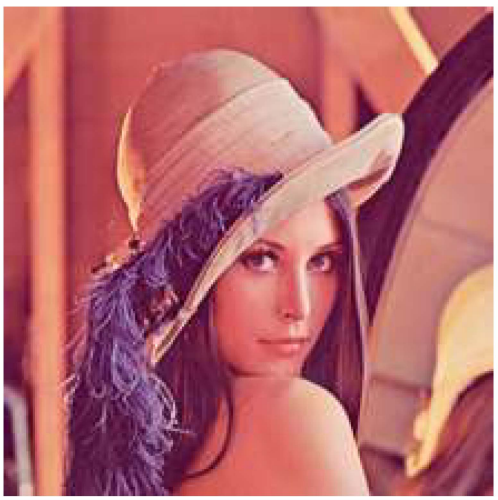
PART A

Binary Image Operations

For binary images logic to be a success we must convert the image to binary by applying a threshold to the image. Look back at previous tutorials to understand how to convert images from greyscale to binary (you may also want to look at the last tutorial to understand what we have to do to convert RGB images to greyscale). Once we have a binary version of our input images we can start to perform some binary logic on them. Download the "mandrill" and "lena" images from Moodle.
These are two common RGB images. For task 1 you are required to evaluate binary image operators on these images.

TASK 1: Following the steps below you are to create a suitable .m file which reads in an image and checks the colour channels on the image. If required then it converts the image.

1) Firstly you create a .py file that reads images and checks the number of channels, if we have an image with 3 channels then we can convert it. For this we use the "np.shape" function,

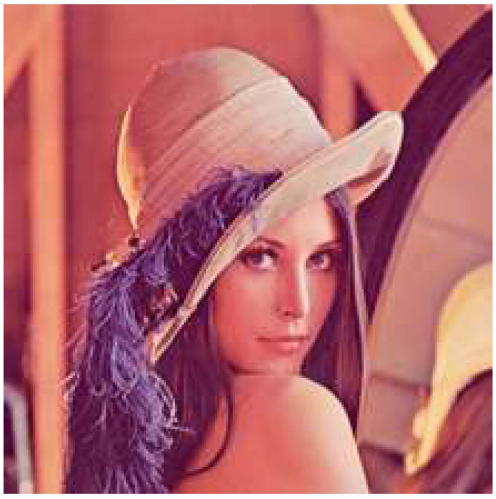


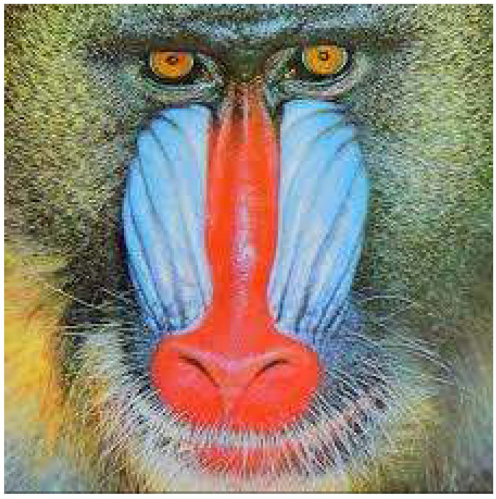
```python
import numpy as np
import cv2

# Load the image using OpenCV
ImageA = cv2.imread('lena.tiff', cv2.IMREAD_UNCHANGED)
# IMREAD_UNCHANGED ensures that the image is loaded as is
#(including alpha channel if it exists)

# Check the dimensions of ImageA and convert if needed
if ImageA is None:
    print("Error: Could not read the image.")
    exit()

dims = np.shape(ImageA)

if len(dims) == 3 and dims[2] == 3:  # RGB image
    Imagea1 = cv2.cvtColor(ImageA, cv2.COLOR_BGR2GRAY)
    print('Image A is an RGB image, it is now converted to grayscale')

 # Check for images with channels > 3 (e.g., RGBA)
elif len(dims) == 3 and dims[2] > 3:
    print('Image A is not an RGB image')
    exit()

else:  # Grayscale image
    print('Image A is a Grayscale image, no conversion needed')
```

NOTE: here we exit the program using the "exit()" function if we have a problem. Therefore if the input image is not of a suitable format we avoid any errors later in processing.

INDEPENDENT TASK 1: Convert your code into a function. Ensure that your function has a suitable name. Ensure also that it takes one argument, the image filename and returns one argument, the image data variable.

Test and evaluate your function and see that it works using the lena, mandrill and pout image and some images of your own choice.



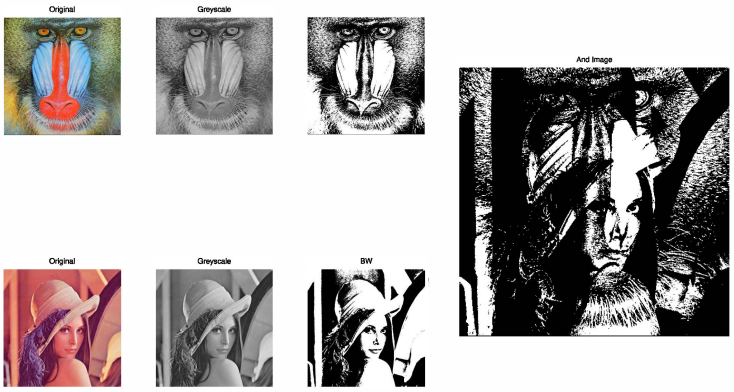


TASK 2: We can now perform a simple check on these two images based on their size. In this example it is important that we have two images of the same x and y size. For other examples this may not be the case. Append your script from independent task 1 to include the code below.

```
%do some simple comparisons of the images to check that
% image sizes are the same.
```

```python
import numpy as np
import cv2

# Load the RGB images
ImageA_RGB = cv2.imread('path_to_imageA.jpg', cv2.IMREAD_COLOR)
ImageB_RGB = cv2.imread('path_to_imageB.jpg', cv2.IMREAD_COLOR)

# Check if the images were successfully loaded
if ImageA_RGB is None or ImageB_RGB is None:
    print("Error: Could not read one or both images.")
    exit()

# Convert the RGB images to grayscale for further processing
ImageA1 = cv2.cvtColor(ImageA_RGB, cv2.COLOR_BGR2GRAY)
ImageB1 = cv2.cvtColor(ImageB_RGB, cv2.COLOR_BGR2GRAY)

# Get the sizes of the images
sizeA = ImageA1.shape
sizeB = ImageB1.shape

if sizeA != sizeB:
    # Resize based on the width and height of ImageA1
    print("The images are different sizes. Resizing ImageB1 to match ImageA1.")
    ImageB1 = cv2.resize(ImageB1, (sizeA[1], sizeA[0]))

else:
    print("The images are the same size, therefore I can continue")

# Now, irrespective of whether they were originally of the same size or not,
#you can proceed to the next steps:
# Threshold the grayscale images to create a binary result
# 127 is the threshold value. You can adjust it if needed.
_, ImageA2 = cv2.threshold(ImageA1, 127, 255, cv2.THRESH_BINARY)
_, ImageB2 = cv2.threshold(ImageB1, 127, 255, cv2.THRESH_BINARY)
```

TASK 3: You can now perform the logical operation on the images using the two arrays of image data and store the result within a new image variable. All resultant images can then be displayed using the subplot environment. Append your script with the following code to complete the plotting.

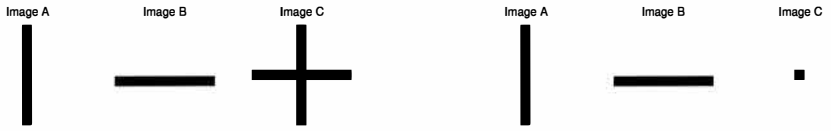



```
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
# Assuming ImageA2 and ImageB2 are already loaded and processed as in the previous code
ImageC = cv2.bitwise_and(ImageA2, ImageB2)

fig = plt.figure()
gs = gridspec.GridSpec(2, 5, width_ratios=[1, 1, 1, 2, 2], height_ratios=[2,1])

ax1 = plt.subplot(gs[0])
ax1.imshow(cv2.cvtColor(ImageA, cv2.COLOR_BGR2RGB))
ax1.set_title('original A')
ax1.axis('off')  # To turn off axis numbers

ax2 = plt.subplot(gs[1])
ax2.imshow(ImageA1, cmap='gray')
ax2.set_title('grayscale')
ax2.axis('off')

ax3 = plt.subplot(gs[2])
ax3.imshow(ImageA2, cmap='gray')
ax3.set_title('Binary')
ax3.axis('off')

ax4 = plt.subplot(gs[5])
ax4.imshow(cv2.cvtColor(ImageB, cv2.COLOR_BGR2RGB))
ax4.set_title('original B')
ax4.axis('off')

ax5 = plt.subplot(gs[6])
ax5.imshow(ImageB1, cmap='gray')
ax5.set_title('grayscale')
ax5.axis('off')

ax6 = plt.subplot(gs[7])
ax6.imshow(ImageB2, cmap='gray')
ax6.set_title('BW')
ax6.axis('off')

# Spanning over multiple positions for the "And Image"
ax7 = plt.subplot(gs[3:5])
ax7.imshow(ImageC, cmap='gray')
ax7.set_title('And Image')
ax7.axis('off')

plt.tight_layout()
plt.show()
```

INDEPENDENT TASK 3:

Append your file to create plots for the `And (&), OR (|)` and `NOT (~)` operators.

INDEPENDENT TASK 4: For the following, which bitwise operator is used with Image A and Image B to create Image C



PART A

Morphology, Erosion and Dilation

Now using sample images we will evaluate the morphological operations of Dilation and Erosion.

For both of these operations to be a success we must define two variables:

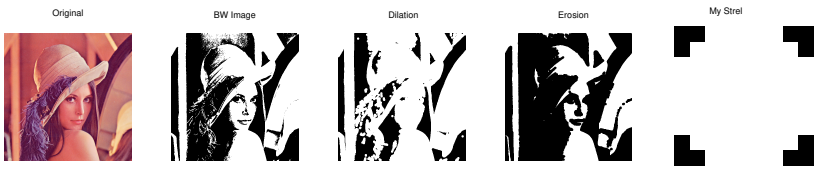
- the input image
- the structure element or shape to perform the morphological operation

This is the shape which is used for the dilation and the erosion. Commonly this shape is referred to as a "Structuring element (or strel)" and can be generated using the "strel" function in Matlab. Once we have these two variables defined we can perform a dilation using the function "imdilate" for Dilation, and "imerode" for Erosion.

TASK 1: Using your files and functions read in the "lena" image, check its dimensions, convert it to grayscale and then convert it to binary. Store this in a variable called "MyImageBW".

TASK 2: Using the code below, create a script (or if you feel adventurous a function) that applies Dilation and then Erosion using this to this image.

```
MyImageBW = np.copy(ImageA2)
# Assuming MyImageBW is already loaded as a grayscale image
# If you have a color image, convert it to grayscale using:
# MyImageBW = cv2.cvtColor(MyImage, cv2.COLOR_BGR2GRAY)

# Create the structuring element (disk with radius 5)
MyStrel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (11, 11))

# Dilation
MyDilation = cv2.dilate(MyImageBW, MyStrel, iterations=1)

# Erosion
MyErosion = cv2.erode(MyImageBW, MyStrel, iterations=1)

# Plotting the images
images = [cv2.cvtColor(ImageA, cv2.COLOR_BGR2RGB), MyImageBW, MyDilation, MyErosion, MyStrel]
titles = ['Original', 'BW Image', 'Dilation', 'Erosion', 'My Strel']

plt.figure()

for i, (img, title) in enumerate(zip(images, titles), 1):
    plt.subplot(1, 5, i)
    if i == 1:  # If it's the original color image
        plt.imshow(img)
    else:
        plt.imshow(img, cmap='gray')
    plt.title(title)
    plt.axis('off')

plt.tight_layout()
plt.show()
```

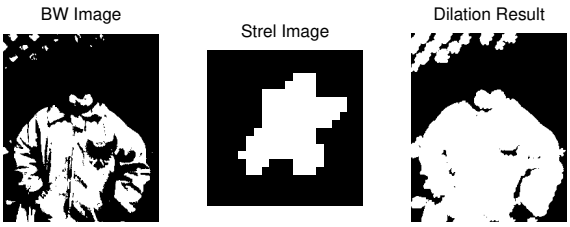


PART B

Images can also be used as the structure element. So for example if we wanted a complex object or a more control over the shape size, then we can create structure element images using Photoshop (available on appsanywhere). However remember that the structure needs to be a binary logical array!

TASK 3:

The example below shows how to use an image called "mystrel.tif" to apply dilation. Create a strel image in Protoshop and apply this code to perform dilation. My example is on the pout image.




```
import cv2
import numpy as np

# Read the structuring element image
my_strel_image = cv2.imread('mystrel.tif', cv2.IMREAD_GRAYSCALE)

# Convert the image data into logical 0-1 data
_, my_strel_image = cv2.threshold(my_strel_image, 127, 255, cv2.THRESH_BINARY)

# Read the main image you want to dilate
my_image_bw = cv2.imread('path_to_your_main_image.jpg', cv2.IMREAD_GRAYSCALE)

# Perform dilation using the structuring element image
my_dilation = cv2.dilate(my_image_bw, my_strel_image, iterations=1)

# Display the result
cv2.imshow('Dilated Image', my_dilation)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

INDEPENDENT TASK I1: Create a function file that performs dilation on an image using another image as the structure element. Name this function something Suitable like "MyDilation" and ensure that it takes both images as input arguments and returns the dilated image.

INDEPENDENT TASK I2: Create a second function file that performs erosion on an image using another image as the structure element. Again name this function something Suitable like "MyErosion" and ensure that it takes both images as input arguments and returns the dilated image. **NOTE: You can use the MyDilation function from above as a starting point.**

INDEPENDENT TASK I3: Using Photoshop, create three different structure elements of different sizes (you can use the images below as an example) and test them on three of your own input images of your choice using your functions from the task above. Try to use Erode to remove unwanted elements from an image and dilate to fill holes, or defects in complete objects.