Coursework 1 – Hybrid Image

Brief

Due date: Friday 11th November, 16:00.
Sample images: hybrid-images.zip in the Coursework folder on Teams
Handin: https://handin.ecs.soton.ac.uk/handin/2223/COMP3204/1/
Required files: `MyConvolution.[java|m|py]`, `MyHybridImages.[java|m|py]`, `[hybridimage.png]`
Credit: 15% of overall module mark

Overview

The goal of this assignment is to write a basic image convolution function and use it to create hybrid images using a simplified version of the SIGGRAPH 2006 paper by Oliva, Torralba, and Schyns (Access via University VPN to download the paper). Hybrid images are static images that change in interpretation as a function of the viewing distance. The basic idea is that high frequency tends to dominate perception when it is available, but, at a distance, only the low frequency (smooth) part of the signal can be seen. By blending the high frequency portion of one image with the low-frequency portion of another, you get a hybrid image that leads to different interpretations at different distances. An example of a hybrid image is shown below.

Example hybrid image. Look at image from very close (or Zoom-in), then from far away (or Zoom-out).

Details

This coursework is intended to familiarise you with image filtering and the implementation of a convolution function in one of Java, Python or Matlab. Once you have created an image convolution function, it is relatively straightforward to construct hybrid images.

Template convolution. Template convolution is a fundamental image processing tool. See section 3.4.1 of Mark's book (Forth Edition) and the lecture materials for more information.

For this assignment, we want you to *hand-code* your own convolution operator using either Java, Python or Matlab, following the instructions below. **You should not make use of any built-in functions or libraries available to you for performing the convolution**.

Your implementation must support arbitrary shaped kernels, as long as both dimensions are odd (e.g. 7x9 kernels but not 4x5 kernels). You should utilise (*possibly implicit*) zero-padding of the input image to ensure that the output image retains the same size as the input image and that the kernel can reach into the image edges and corners. The implementation must also support convolution of both grey-scale and colour images. Note that colour convolution is achieved by applying the convolution operator to each of the colour bands separately (i.e. treating each band as an independent grey-level image).

Make sure that you **implement the convolution operator and not a different (but similar) operator**. Check that your implementation works correctly for non-symmetric kernels. You can try to implement the convolution using the Fourier transform; however please note that the specification asks for **zero-padding** of the image. The Fourier transform replicates to infinity by virtue of the sampling process, so you will have to make allowances for this with appropriate padding and cropping.

Specific instructions on how your function should be written.

<Java>
If you choose Java, you're going to use the [OpenIMAJ](http://openimaj.org) library as the basis for manipulating images. You will need to have worked through
[Chapter 1] (http://www.openimaj.org/tutorial/getting-started-with-openimaj-using-maven.html),
[Chapter 2](http://www.openimaj.org/tutorial/processing-your-first-image.html)
and [Chapter 7](http://www.openimaj.org/tutorial/processing-video.html)
of the [OpenIMAJ tutorial](http://www.openimaj.org/tutorial/) prior to starting this coursework.

OpenIMAJ has numerous built in and highly efficient operators to perform convolution, but you will be writing your own such function from scratch for this assignment. More specifically, you will implement a class called `MyConvolution` that builds on this skeleton:

```
package uk.ac.soton.ecs.<your_username>.hybridimages;

import org.openimaj.image.FImage;
import org.openimaj.image.processor.SinglebandImageProcessor;

public class MyConvolution implements SinglebandImageProcessor<Float, FImage> {
	private float[][] kernel;

	public MyConvolution(float[][] kernel) {
		//note that like the image pixels kernel is indexed by [row][column]
		this.kernel = kernel;
	}

	@Override
	public void processImage(FImage image) {
		// convolve image with kernel and store result back in image
		//
		// hint: use FImage#internalAssign(FImage) to set the contents
		// of your temporary buffer image to the image.
	}
}
```

You will need to fill in the `processImage` method so that it performs convolution of the image with the kernel/template. Note that the code you write for template convolution is designed to work on grey-level images (`FImage`), however the images you will process in the next section are colour (`MBFImage`). Convolution of a colour image will be performed by separately convolving each of the colour bands with the same kernel. OpenIMAJ will automatically take care of this for you when you pass your `MyConvolution` instance to the `process` method of an `MBFImage`.

<Python>

If you choose to implement the assignment in Python, you'll need to implement the convolution function in a file called `MyConvolution.py` using the following code skeleton:

```
import numpy as np


def convolve(image: np.ndarray, kernel: np.ndarray) -> np.ndarray:
    """
    Convolve an image with a kernel assuming zero-padding of the image to handle the borders

    :param image: the image (either greyscale shape=(rows,cols) or colour shape=(rows,cols,channels))
    :type numpy.ndarray

    :param kernel: the kernel (shape=(kheight,kwidth); both dimensions odd)
    :type numpy.ndarray

    :returns the convolved image (of the same shape as the input image)
    :rtype numpy.ndarray
    """
    # Your code here. You'll need to vectorise your implementation to ensure it runs
    # at a reasonable speed.
```

As you can see, the images are represented as `numpy` arrays (either 2d or 3d for greyscale and colour, respectively). You will need to vectorise your implementation to make it run at a sensible speed (even with a 3x3 kernel a naive implementation using four nested loops will be *very* slow, whereas an implementation with the two inner loops vectorised will be significantly faster).

Restrictions

You must not use any built-in or library functions for implementing the convolution, except for basic math/matrix commands like `np.sum`, and `np.zeros`. Do not put any other `import` statements in your file.

<Matlab>

If you choose to implement the assigment in Matlab, you'll need to implement the convolution function in a file called `myconvolution.m` using the following code skeleton:

```
function convolved=myconvolution(image, kernel)
% MYCONVOLUTION  Convolve an image with a kernel
%   C = MYCONVOLUTION(image,kernel) returns the two-dimensional convolution
%       of matrices image and kernel assuming zero-padding of the image to handle the borders.
%
%   image should have size (rows,cols) or (rows,cols,channels)
%   kernel should have size (krows,kcols); both dimensions odd

% YOUR CODE HERE
```

As you can see, the images are represented as matrices (either 2D or 3D for greyscale and colour, repectively). You probably want to consider vectorising the two inner loops of your implementation to make it run at a sensible speed with large kernels (needed for the 2nd part of the assignment).

Restrictions

You must not use any built-in or library functions for implementing the convolution, except for basic math/matrix commands like `mod`, `sum`, `floor`, and `zeros`. For example, use of the following Matlab functions are forbidden: `imfilter()`, `filter2()`, `conv2()`, `nlfilter()`, `colfilt()`.

Hybrid Images

A hybrid image is the sum of a low-pass filtered version of the one image and a high-pass filtered version of a second image. There is a free parameter, which can be tuned for each image pair, which controls *how much* high frequency to remove from the first image and how much low frequency to leave in the second image. This is called the "cutoff-frequency". In the paper it is suggested to use two cutoff-frequencies (one tuned for each image) and you are free to try that, as well.

Low pass filtering (removing all the high frequencies) can be achieved by convolving the image with a Gaussian filter. The cutoff-frequency is controlled by changing the standard deviation, sigma, of the Gaussian filter used in constructing the hybrid images. You will need to implement a function to generate a 2D Gaussian convolution kernel of `size*size` pixels. The `size` is width and height of the filter in pixels. It is standard practice for this to be set as a function of the sigma value as follows:

```
int size = (int) (8.0f * sigma + 1.0f); // (this implies the window is +/- 4 sigmas from the centre of
the Gaussian)
if (size % 2 == 0) size++; // size must be odd
```

High pass filtering (removing all the low frequencies) can be most easily achieved by subtracting a low-pass version of an image from itself.

You should implement your hybrid images functionality by completing the code skeleton for your chosen language (you must use the same language for both parts of the assignment!).

<Java>
package uk.ac.soton.ecs..hybridimages;

```
import org.openimaj.image.MBFImage;

public class MyHybridImages {
	/**
	 * Compute a hybrid image combining low-pass and high-pass filtered images
	 *
	 * @param lowImage
	 *            the image to which apply the low pass filter
	 * @param lowSigma
	 *            the standard deviation of the low-pass filter
	 * @param highImage
	 *            the image to which apply the high pass filter
	 * @param highSigma
	 *            the standard deviation of the low-pass component of computing the
	 *            high-pass filtered image
	 * @return the computed hybrid image
	 */
	public static MBFImage makeHybrid(MBFImage lowImage, float lowSigma, MBFImage highImage,
float highSigma) {
		//implement your hybrid images functionality here.
		//Your submitted code must contain this method, but you can add
		//additional static methods or implement the functionality through
		//instance methods on the `MyHybridImages` class of which you can create
		//an instance of here if you so wish.
		//Note that the input images are expected to have the same size, and the output
		//image will also have the same height & width as the inputs.
	}

	public static float[][] makeGaussianKernel(float sigma) {
		//Use this function to create a 2D gaussian kernel with standard deviation sigma.
		//The kernel values should sum to 1.0, and the size should be floor(8*sigma+1) or
		//floor(8*sigma+1)+1 (whichever is odd) as per the assignment specification.
	}
}
```

Restrictions

You must use your `MyConvolution` class to perform the convolutions in your `makeHybrid` method. Do not attempt to import or use the OpenIMAJ convolution functions as this will fail during the automatic marking of your code!

Testing your code

You will need to thoroughly test your implementation, paying attention to each of the three component parts (convolution, gaussian kernel creation and hybrid creation). You will need to implement your own test harness; this should be in a separate file that will not form part of the assessment of the coursework (but is vital to you having a correct implementation!).

We have provided a tool in the `SubmissionTester.jar` jar file (downloadable in Teams) that is capable of performing some elementary tests on the java code that you will submit (like checking that it compiles, that the `MyConvolution` class can be instantiated and that the required methods run. You can run the tool from the commandline (you'll need java>=1.8.0) using:

```
java -jar SubmissionTester.jar path/to/MyConvolution.java path/to/MyHybridImages.java
```

The tool doesn't perform any tests to check your code actually works correctly however, so you should check this yourself before submission of the files to handin! Note that when your code runs, it is executed in

a restricted sandbox environment, and will throw errors if you try to read or write files, or access the network.

<Python>
In a file called `MyHybridImages.py`:

```
import math
import numpy as np

from MyConvolution import convolve


def myHybridImages(lowImage: np.ndarray, lowSigma: float, highImage: np.ndarray, highSigma:
float) -> np.ndarray:
    """
    Create hybrid images by combining a low-pass and high-pass filtered pair.

    :param lowImage: the image to low-pass filter (either greyscale shape=(rows,cols) or
colour shape=(rows,cols,channels))
    :type numpy.ndarray

    :param lowSigma: the standard deviation of the Gaussian used for low-pass filtering
lowImage
    :type float

    :param highImage: the image to high-pass filter (either greyscale shape=(rows,cols) or
colour shape=(rows,cols,channels))
    :type numpy.ndarray

    :param highSigma: the standard deviation of the Gaussian used for low-pass filtering
highImage before subtraction to create the high-pass filtered image
    :type float

    :returns returns the hybrid image created
           by low-pass filtering lowImage with a Gaussian of s.d. lowSigma and combining it
with
           a high-pass image created by subtracting highImage from highImage convolved with
           a Gaussian of s.d. highSigma. The resultant image has the same size as the input
images.
    :rtype numpy.ndarray
    """

    # Your code here.


def makeGaussianKernel(sigma: float) -> np.ndarray:
    """
    Use this function to create a 2D gaussian kernel with standard deviation sigma.
    The kernel values should sum to 1.0, and the size should be floor(8*sigma+1) or
    floor(8*sigma+1)+1 (whichever is odd) as per the assignment specification.
    """

    # Your code here.
```

Restrictions

You must use your `MyConvolution.convolve` function to perform the convolutions in your `myHybridImages` function. Do not put any other `import` statements in your file. You can use basic math/matrix functions like `np.zeros`, `np.sum`, etc., as required.

Testing your code

You will need to thoroughly test your implementation, paying attention to each of the three component parts (convolution, gaussian kernel creation and hybrid creation). You will need to implement your own test harness; this should be in a separate file that will not form part of the assessment of the coursework (but is vital to you having a correct implementation!).

<Matlab>
In a file called `myhybridimages.m`:

```
function hybrid=myhybridimages(lowImage, lowSigma, highImage, highSigma)
% MYHYBRIDIMAGES  Create hybrid images by combining a low-pass and high-pass filtered pair.
%   C = MYHYBRIDIMAGES(lowImage, lowSigma, highImage, highSigma) returns the hybrid image
created
%       by low-pass filtering lowImage with a Gaussian of s.d. lowSigma and combining it
with
%       a high-pass image created by subtracting highImage from highImage convolved with
%       a Gaussian of s.d. highSigma
%
%   lowImage and highImage should both have size (rows,cols) or (rows,cols,channels).
%   The resultant image also has the same size

% YOUR CODE HERE
end

function f=makegaussiankernel(sigma)
% Use this function to create a 2D gaussian kernel with standard deviation sigma.
% The kernel values should sum to 1.0, and the size should be floor(8*sigma+1) or
% floor(8*sigma+1)+1 (whichever is odd) as per the assignment specification.

% YOUR CODE HERE
end
```

Restrictions

You must use your `myconvolution` function to perform the convolutions in your `myhybridimages` function. You must not use any built-in or library functions, except for basic math/matrix commands like `mod`, `sum`, `floor`, and `zeros`.

Testing your code

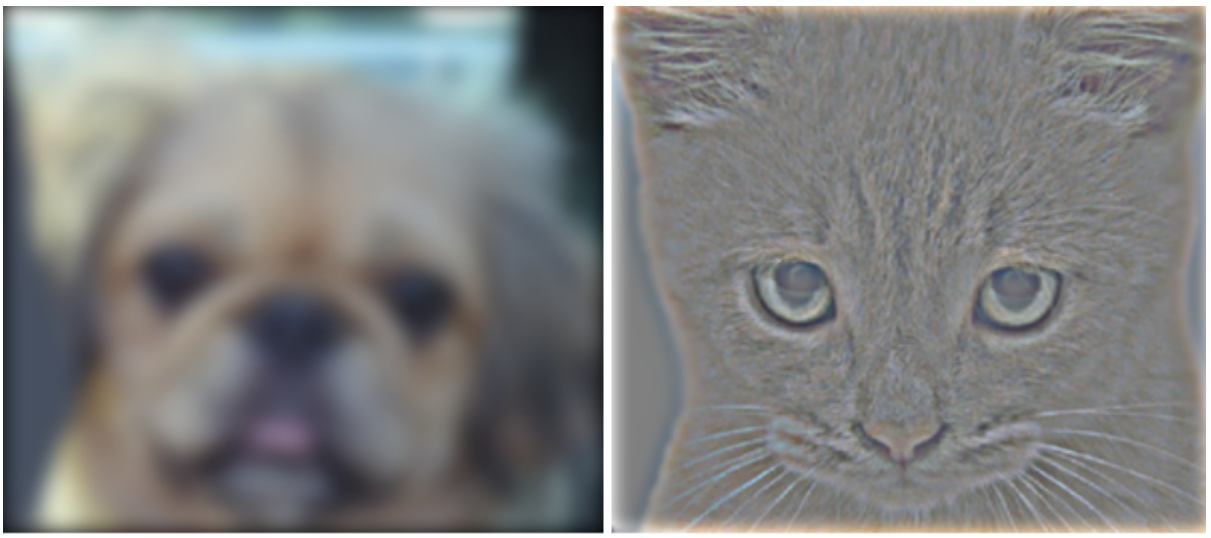
You will need to thoroughly test your implementation, paying attention to each of the three component parts (convolution, gaussian kernel creation and hybrid creation). You will need to implement your own test harness; this should be in a separate file that will not form part of the assessment of the coursework (but is vital to you having a correct implementation!).

Image set

We have provided you with 5 pairs of aligned images in the "`hybrid-images.zip`" file which can be merged reasonably well into hybrid images. The alignment is important because it affects the perceptual grouping (read the paper for details). We encourage you to create an additional example (e.g. change of expression, morph between different objects, change over time, etc.) for full marks (see below). See the hybrid images project page for some inspiration.

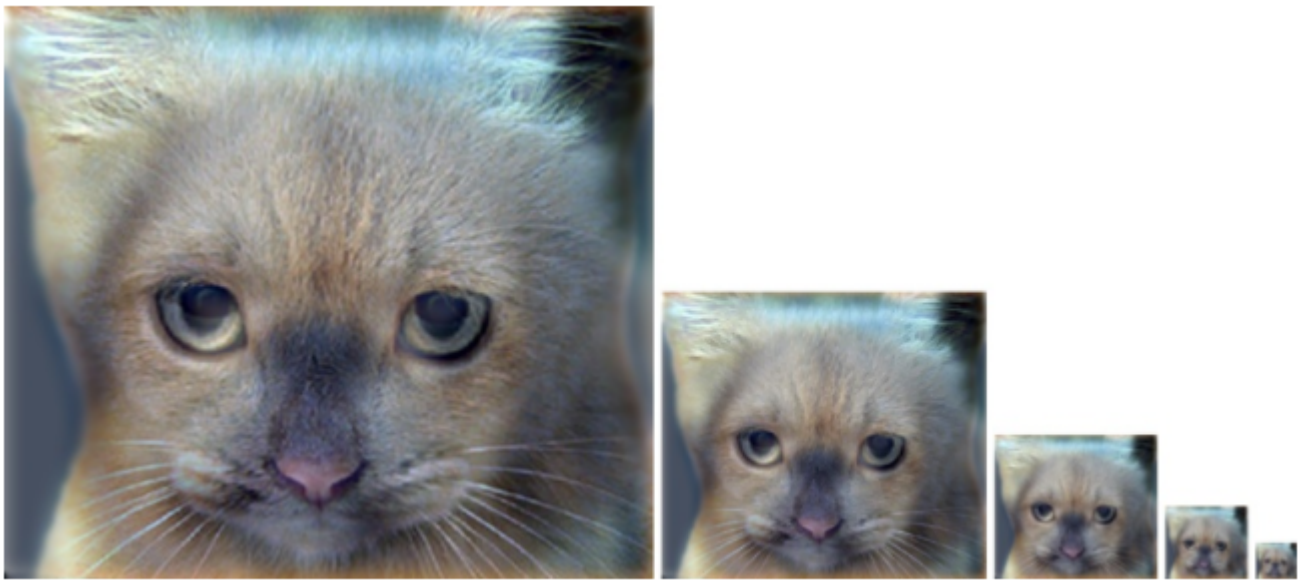
For the example shown at the top of the page, the two original images look like this:

The low-pass (blurred) and high-pass versions of these images look like this:



The high frequency image is actually zero-mean with negative values so it is visualised by adding 0.5 (128 in 255 dynamic range) to every pixel in each colour channel. In the resulting visualisation, bright values are positive and dark values are negative.

Adding the high and low frequencies together gives you the image at the top of this page. If you're having trouble seeing the multiple interpretations of the image, a useful way to visualise the effect is by progressively down-sampling the hybrid image as is done below:



What to hand in

You are required to submit the following items to ECS Handin:

- Your `MyConvolution.[java|m|py]` file
- Your `MyHybridImages.[java|m|py]` file

For full marks, you also need to submit a hybrid image creation of your own (ideally with the progressive downsampling shown above). Details below.

Marking and feedback

You will receive a grade out of 10 for this coursework; this will be scaled to be out of 15 for the 100 marks available for the overall module.

This coursework can be primarily automatically marked by a program that compiles/interprets and runs your submitted files with a number of different parameters. We then provide a grade out of 8 (split 4/4 between the convolution and hybrid images parts), with brief individual feedback (the general feedback will be given in the lecture). The remaining 2 marks are available if you upload a novel hybrid image of your own creation. We're looking for an image which clearly encapsulates and demonstrates the notions of a hybrid image formation with particular demonstration of the effects consistent with progressive down-sampling. Full marks will only be awarded for images that are particularly creative, impressive or funny. A selection of the best images will be shown to the class during a feedback lecture, which will cover a broad range of lessons related to the coursework.

Standard ECS late submission penalties apply.

Useful links

- SIGGRAPH Hybrid Images Paper
- The Hybrid Images project page
- **Using Matlab:**
  - Image processing toolbox tutorials
- **Libraries for Java programmers**
  - OpenIMAJ

Questions

If you have any problems/questions, use the Q&A channel on Teams, or email Xiaohao and Hansung.