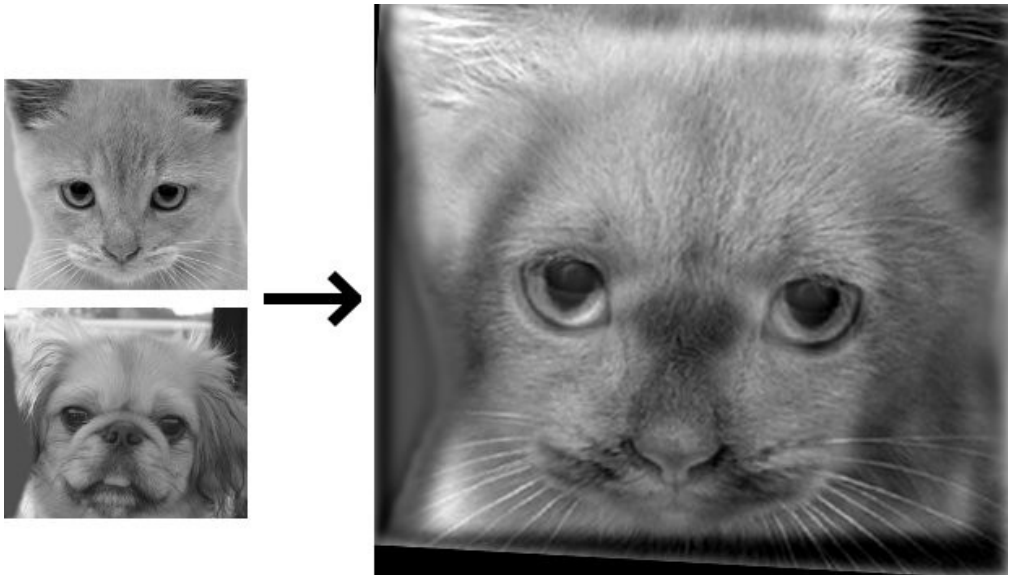
Homework #2

Image Pyramids and Template Matching

Please present a report with all your answers, explanations, and sample images or plots. Submit also a soft copy of the source code and binaries used to generate these results. Please note that copying of any results or source code will result in ZERO credit for the whole homework.

Part 1: Hybrid Images



Write a program/script that takes in two images I_0 and I_1 and produces their hybrid image I_h i.e. a composite image such that I_0 is visible at longer distances and I_1 is viewable at shorter distances.

The basic idea is to add a low-pass filtered version of I_0 and a high-pass filtered version of I_1 together, so that the smoothed image appears at longer distances and the sharp image appears at shorter distances.

$$I_h = G * I_0 + (1 - G) * I_1$$

where *G* is a Gaussian filter. See this paper by Oliva and Torralba for more details.

To implement this you need to set the cut-off frequencies of the LPF and the HPF properly so that the blending is correct. This can be done by constructing a Gaussian pyramid and a Laplacian pyramid for both images, and choosing only the top/bottom *N* levels in each pyramid, adding them and then

reconstructing the image from the pyramid.

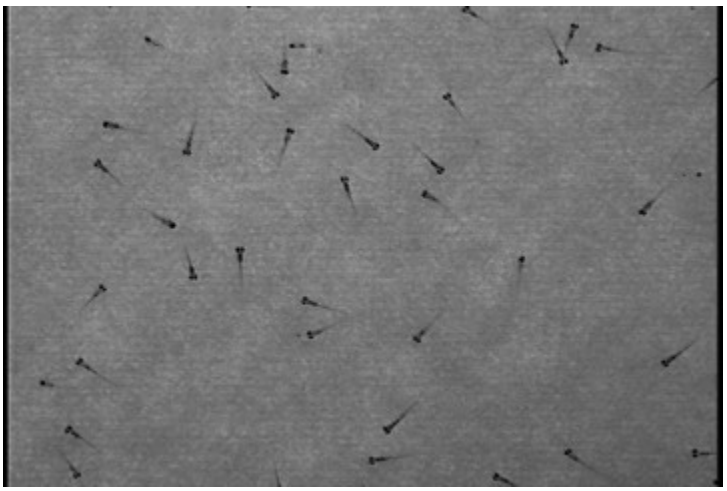
The flow should be as follows:

1. Build a Gaussian and a Laplacian pyramid for each images.
2. Choose the cut-off frequency (the pyramid levels) for the Gaussian pyramid of the LPF image I_0 and for the Laplacian pyramid of the HPF image I_1.
3. Reconstruct each image from its Laplacian pyramid starting at the cut-off level.
4. Add the final reconstructed and filtered image to produce the hybrid image I_h.

Required

1. Working implementation of the image pyramid construction and reconstruction. You can use any functions that do convolution, filtering, up/downsampling, … but ***not*** image pyramid construction.
2. Working implementation of hybrid image generation.
3. Three sample hybrid images of your choice.

Part 2: Template Matching



In this part you will implement a template matching algorithm to look for a template inside an image. The template will be an image of a "*zebra fish*" and the goal is to find *all* instances of the fish in an input image containing many of them. Template matching works by passing the template over all locations in the image, computing a distance(similarity) measure at each location. Those locations with local minima(maxima) denote locations where the template exists in the image.

1. Look at images *zebra-03.png*, *zebra-04.png*, and *zebra-05.png* and choose a patch that is good to represent the zebra fish. Implement a template matching algorithm to look for this patch in *zebra-01.png* and *zebra-02.png*. Try the **Zero-Mean Correlation** and **Sum of Absolute Differences (SAD)** distance measures and compare their results. Which one is faster? Which one finds more fish? Which one do you prefer? Why?
2. Since the individual fish are at different orientations, we need to modify the algorithm above to look for fish at different rotation angles. Explain one way to do that and implement it with the better distance measure from the first part. How many fish can you find in *zebra-01.png* and *zebra-02.png*?
3. Now that we solved the rotation problem, we need to solve the scale problem. Extract a patch from either *zebra-01-s.png* or *zebra-02-s.png,* and explain how can you use this patch to search images *zebra-01.png*, *zebra-01-m.png*, *zebra-02.png*, and *zebra-02-m.png.* Implement your idea and count how many fish you can find.

Acknowledgments

Part 1 is adapted from CS134 at Brown by Prof. James Hays, and Part 2 is adapted from EE148 at Caltech by Prof. Pietro Perona.