

Project 1 (A) Coding: Hybrid Images

CS 4670/5670, Spring 2016 - Intro to Computer Vision



Look at the image from very close and then very far.

Key Information

Assigned	Friday, February 5th 2016 (Code is on CMS)
Due	Friday, February 12th 2016 by 1:00pm on CMS
Artifact Due	Sunday, February 14st by 2016 1:00pm on CMS

Note: This assignment will be autograded, with a focus on efficiency of code. Even though you are not supposed to use built-in OpenCV, Numpy, or Scipy functions, or a third-party library, you are required to familiarize yourself with vectorizing Numpy code. There will be a lecture dedicated to Numpy and vectorization on Feb 8, 2016. **Your code will be given two minutes to run** on the course VM, on a 4.0 GHz Intel x86_64 machine, with 32 GB of RAM for a single iteration.

Both PA 1 (A) and (B) must be done in groups of 2.

Overview

The goal of this assignment is to write an image filtering function and use it to create [hybrid images](#) using a simplified version of the SIGGRAPH 2006 [paper](#) by Oliva, Torralba, and Schyns. 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, i.e., when viewed closeby. 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.

You will use your own solution to create your own hybrid images that you submit. The class will vote on the best hybrid image created.

In the **Downloads** section below, we provide you with a virtual machine that has all the necessary dependencies installed for you to run this project. Apart from that, there is skeleton code for a user interface provided on [CMS](#) along with a file **hybrid.py** that contains functions that you need to implement. We will walk you through the functions in the next section. If you have issues running the VM or the user interface, please post a question on Piazza, or visit a TA during office hours.

Implementation Details

This project is intended to familiarize you with Python, NumPy and image filtering. Once you have created an image filtering function, it is relatively straightforward to construct hybrid images.

This project requires you to implement 5 functions each of which builds onto a previous function:

1. **cross_correlation_2d**
2. **convolve_2d**
3. **gaussian_blur_kernel_2d**
4. **low_pass**
5. **high_pass**

Image Filtering. Image filtering (or convolution) is a fundamental image processing tool. See chapter 3.2 of Szeliski and the lecture materials to learn about image filtering (specifically linear filtering). Numpy has numerous built in and efficient functions to perform image filtering, but you will be writing your own such function from scratch for this assignment. More specifically, you will implement **cross_correlation_2d**, followed by **convolve_2d** which uses **cross_correlation_2d**.

Gaussian Blur

As you have seen in the lectures, there are a few different ways to blur an image, for example taking an unweighted average of the neighboring pixels. Gaussian blur is a special kind of *weighted* averaging of neighboring pixels, and is described in the lecture slides. To implement Gaussian blur, you will implement a function **gaussian_blur_kernel_2d** that produces a kernel of a given *height* and *width* which can then be passed to **convolve_2d** from above, along with an image, to produce a blurred version of the image.

High and Low Pass Filters

Recall that a low pass filter is one that removed the fine details from an image (or, really, any *signal*), whereas a high pass filter only retains the fine details, and gets rid of the coarse details from an image. Thus, using Gaussian blurring as described above, implement **high_pass** and **low_pass** functions.

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. In the starter code, the cutoff frequency is controlled by changing the standard deviation (sigma) of the Gaussian filter used in constructing the hybrid images. We provide you with the code for creating a hybrid image, using the functions described above.

Forbidden functions

For just this assignment (PA1-part A), you should **NOT** use any Numpy, Scipy, OpenCV, or other pre-implemented functions for filtering. This limitation will be lifted in future assignments, but for now, you should use for loops or Numpy vectorization to apply a kernel to each pixel in the image. The bulk of your code will be in **cross_correlation_2d**, and **gaussian_blur_kernel_2d** with the other functions using these functions either directly or through one of the other functions you implement.

We have provided a GUI in **gui.py**, to help you debug your image filtering algorithm. To see a pre-labeled version of the sample images run:

```
python gui.py -t resources/sample-correspondance.json -c resources/sample-config.json
```

We provide you with a pair of images that need to be aligned using the GUI. The code for alignment uses an affine transform to map the eyes to eyes and nose to nose, etc. as you

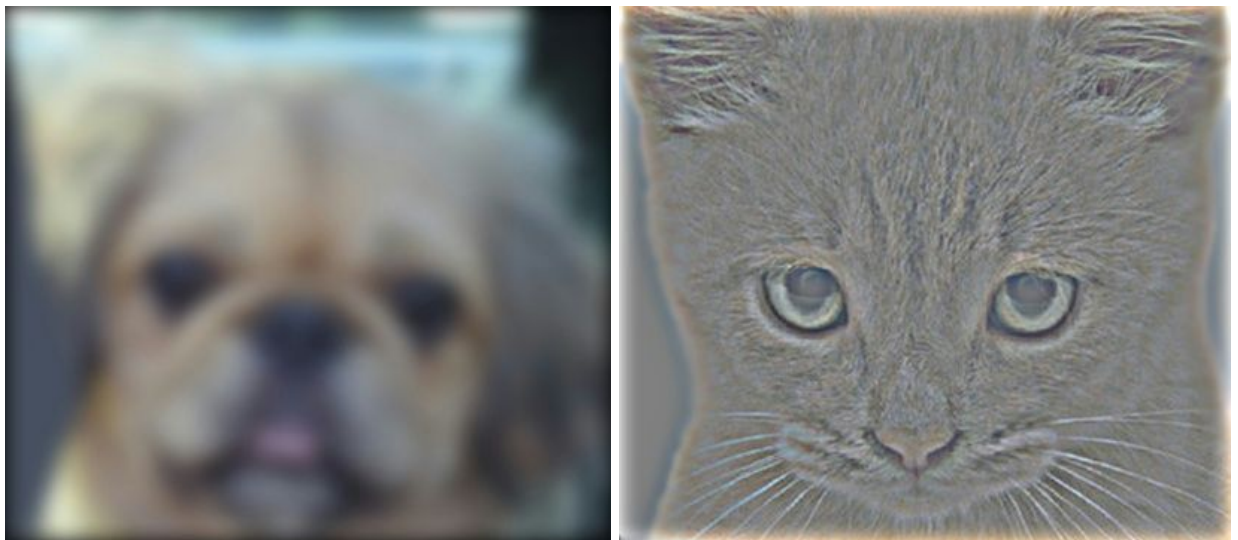
specify on the UI. We encourage you to create additional examples (e.g., change of expression, morph between different objects, change over time, etc.). See the [hybrid images project page](#) for some inspiration. The project page also contains materials from their [Siggraph presentation](#).

Examples

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:



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 visualize the effect is by progressively downsampling the hybrid image as is done below:



Downloads

- Course [Virtual Machine \(VM\)](#)
- [Virtualbox](#) for running the VM
- Skeleton code is on CMS
- [Video tutorial](#) on using the project user interface

Credits

Assignment based on versions developed by James Hays and Derek Hoiem.