Project 1: Image Filtering

CS 6476

Fall 2024

Logistics

- Due: Check Canvas for up to date information.
- Hand-in: Gradescope

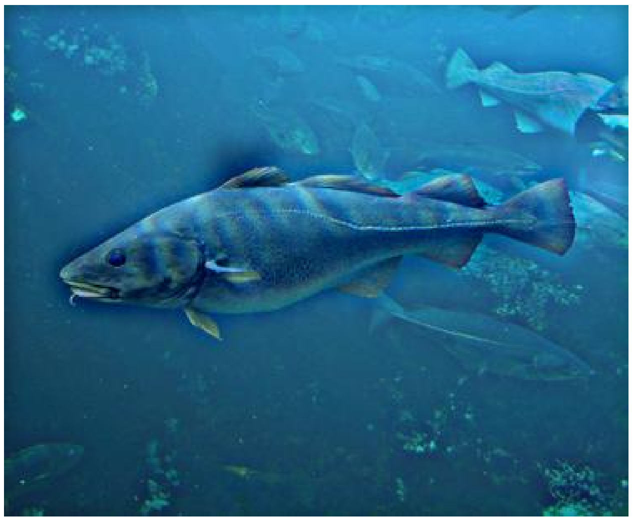


Figure 1: Look at the image from very close, then from far away.

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 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.

This project is intended to familiarize you with `Python`, `PyTorch`, and image filtering. Once you have created an image filtering function, it is relatively straightforward to construct hybrid images. If you don't already know Python, you may find this resource helpful. If you are more familiar with MATLAB, this guide is very helpful. If you're unfamiliar with PyTorch, the tutorials from the official website are useful.

Setup

1. Use `conda/install.sh` for environment installation.
2. Run the notebook using `jupyter notebook ./project-1.ipynb`
3. After implementing all functions, ensure that all sanity checks are passing by running `pytest tests` inside the main folder.
4. Generate the zip folder for the code portion of your submission once you've finished the project using `python zip_submission.py --gt_username <your_gt_username>`

1 NumPy

1.1 Gaussian Kernels

Univariate Gaussian Kernels. Gaussian filters are used for blurring images. You will first implement `create_Gaussian_kernel_1D()`, a function that creates a 1D Gaussian vector according to two parameters: the kernel size (length of the 1D vector) and σ, the standard deviation of the Gaussian. The vector should have values populated from evaluating the 1D Gaussian probability density function (pdf) at each coordinate. The 1D Gaussian pdf is defined as:

$$f_X(x;\mu,\sigma^2) \triangleq \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{1}{2\sigma^2}(x-\mu)^2\right) \tag{1}$$

Multivariate Gaussian Kernels. Next, you will implement `create_Gaussian_kernel_2D()`, which creates a 2-dimensional Gaussian kernel according to a free parameter, *cutoff frequency*, which controls how much low frequency to leave in the image. Choosing an appropriate cutoff frequency value is an important step for later in the project when you create hybrid images. We recommend that you implement `create_Gaussian_kernel_2D()` by creating a 2D Gaussian kernel as the outer product of two 1D Gaussians, which you have now already implemented in `create_Gaussian_kernel_1D()`. This is possible because the 2D Gaussian filter is *separable* (think about how $e^{(x+y)} = e^x \cdot e^y$). The multivariate Gaussian function is defined as:

$$f_{\boldsymbol{X}}(\boldsymbol{x};\boldsymbol{\mu},\boldsymbol{\Sigma}) \triangleq \frac{1}{Z}\exp\left(-\frac{1}{2}(x-\boldsymbol{\mu})^\top\boldsymbol{\Sigma}^{-1}(x-\boldsymbol{\mu})\right) \tag{2}$$

where $\boldsymbol{\mu}\in\mathbb{R}^2$ is the mean vector, $\boldsymbol{\Sigma}\in\mathbb{R}^{2\times 2}$ is the covariance, and Z is the normalization constant:

$$Z \triangleq \frac{1}{(2\pi)^{n/2}\det(\boldsymbol{\Sigma})^{1/2}} \tag{3}$$

Properties of Gaussian Kernels. It is a fact that for the Gaussian pdf:

$$\boldsymbol{\mu} = \underset{\boldsymbol{x}}{\operatorname{argmax}}\ f_{\boldsymbol{X}}(\boldsymbol{x};\mu,\Sigma) \tag{4}$$

Indeed, for discrete Gaussian kernels which are indexed from 0 to $k-1$, it is the case that: $\boldsymbol{\mu} = \left(\lfloor\frac{k}{2}\rfloor, \lfloor\frac{k}{2}\rfloor, \ldots, \lfloor\frac{k}{2}\rfloor\right)$ – this is shown qualitatively in fig. 2. Additionally, we can see that Z is defined as such to ensure that the distribution in eq. (2) satisfies the second axiom of probability:

$$\int_{-\infty}^{\infty} f_{\boldsymbol{X}}(\boldsymbol{x};\boldsymbol{\mu},\boldsymbol{\Sigma}) = 1 \tag{5}$$

It should be noted that if your kernel (discrete) does not satisfy the above axiom, you should normalize it with:

$$Z = \sum_{i=0}^{k-1}\sum_{j=0}^{k-1} f_{\boldsymbol{X}}((i,j);\boldsymbol{\mu},\boldsymbol{\Sigma}) \tag{6}$$

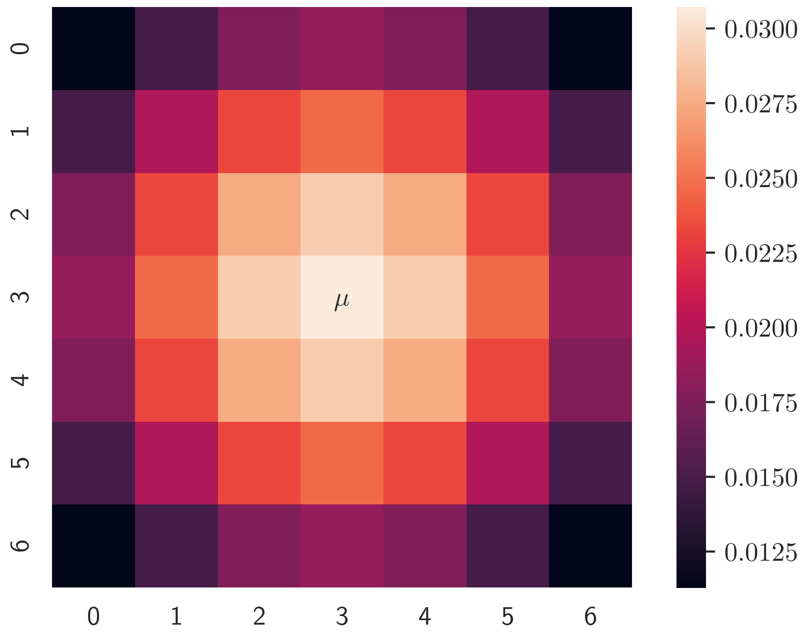


Figure 2: A 7×7 Gaussian kernel with $\boldsymbol{\mu} = (\lfloor \frac{7}{2} \rfloor, \lfloor \frac{7}{2} \rfloor)$ and $\boldsymbol{\Sigma} = I$.

1.2 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). You will be writing your own function to implement image filtering from scratch. More specifically, you will implement `my_conv2d_numpy()` which imitates the `filter2D()` function in the OpenCV library. As specified in `part1.py`, your filtering algorithm must: (1) support grayscale and color images, (2) support arbitrarily-shaped filters, as long as both dimensions are odd (e.g., 7×9 filters, but not 4×5 filters), (3) pad the input image with zeros, and (4) return a filtered image which is the same resolution as the input image. We have provided an iPython notebook, `project-1.ipynb` and some unit tests (which are called in the notebook) to help you debug your image filtering algorithm. Note that there is a time limit of 5 minutes for a single call to `my_conv2d_numpy()`, so try to optimize your implementation if it goes over.

1.3 Hybrid Images

A hybrid image is the sum of a low-pass filtered version of one image and a high-pass filtered version of another image. As mentioned above, *cutoff frequency* controls how much high frequency to leave in one image and how much low frequency to leave in the other image. In `cutoff_frequencies.txt`, we provide a default value of 7 for each pair of images (the value of line i corresponds to the cutoff frequency value for the i-th image pair). You should replace these values with the ones you find work best for each image pair. 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 of the Gaussian filter used in constructing the hybrid images. You will first implement `create_hybrid_image()` according to the starter code in `part1.py`. Your function will call `my_conv2d_numpy()` using the kernel generated from `create_Gaussian_kernel()` to create low and high frequency images, and then combine them into a hybrid image.

2 PyTorch

You will now implement creating hybrid images again but using PyTorch.

First, you need to prepare your dataset. The `HybridImageDataset` class will create tuples using pairs of images with a corresponding cutoff frequency (which you should have found from experimenting in Part 1). The image paths will be loaded from `data/` using `make_dataset()` and the cutoff frequencies from `cutoff_frequencies.txt` using `get_cutoff_frequencies()`. Additionally, you will implement `__len__()`, which returns the number of image pairs, and `__getitem__()`, which returns the `i`-th tuple. Refer to this tutorial for additional information on data loading & processing.

Next, you will implement the `HybridImageModel` class. Instead of using your implementation of `my_conv2d_numpy()` to get the low and high frequencies from a pair of images, `low_pass()` should use the 2D convolution operator from `torch.nn.functional` to apply a low pass filter to a given image. You will have to implement `get_kernel()` which calls your `create_Gaussian_kernel()` function from `part1.py` for each pair of images using the cutoff frequencies as specified in `cutoff_frequencies.txt`, and reshape it to the appropriate dimensions for PyTorch. Then, similar to `create_hybrid_image()` from `part1.py`, `forward()` will call `get_kernel()` and `low_pass()` to create the low and high frequency images, and combine them into a hybrid image. Refer to this tutorial for additional information on defining neural networks using PyTorch.

You will compare the runtimes of your hybrid image implementations from Parts 1 & 2.

3 Understanding input/output shapes in PyTorch

You will now implement `my_conv2d_pytorch()` in `part3.py` using the same 2D convolution operator from `torch.nn.functional` used in `low_pass()`.

Before we proceed, here are three quick definitions of terms we'll use often when describing convolution:

- **Stride**: When the stride is 1 then we move the filters one pixel at a time. When the stride is 2 (or uncommonly 3 or more, though this is rare in practice) then the filters jump 2 pixels at a time as we slide them around.
- **Padding**: The amount of pixels added to an image when it is being convolved with a the kernel. Padding can help prevent an image from shrinking during the convolution operation.
- **Dilation**: In a dilated convolution, the convolution kernel is applied over an area larger than its length by skipping input values with the dilation rate. For example, a dilation rate of 1 means a standard convolution, while a dilation rate of 2 means that the kernel skips every other input element. This allows the network to capture information from a broader context while preserving spatial resolution.

Unlike `my_conv2d_numpy()` from `part1.py`, the shape of your output does not necessarily have to be the same as the input image. Instead, given an input image of shape $(1, d_1, h_1, w_1)$ and kernel of shape $(N, \frac{d_1}{g}, k, k)$, your output will be of shape $(1, d_2, h_2, w_2)$ where g is the number of groups, $k' = k * d - d + 1$, $d_2 = N$, $h_2 = \frac{h_1 - k' + 2 * p}{s} + 1$, and $w_2 = \frac{w_1 - k' + 2 * p}{s} + 1$, and p, s, d are padding, stride and dilation, respectively. Refer to this online website for help.

Think about *why* the equations for output width w_2 and output height h_2 are true – try sketching out a 5×5 grid, and seeing how many places you can place a 3×3 square within the grid with stride 1. What about with stride 2? Does your finding match what the equation states?

We demonstrate the effect of the value of the `groups` parameter on a simple example with an input image of shape $(1, 2, 3, 3)$ and a kernel of shape $(4, 1, 3, 3)$:

4 Image Sharpening using Fourier Transform

In this part, you will enhance the sharpness of an image using a known Laplacian kernel in the frequency domain. We have discussed in class the relationship between convolution in the spatial domain and multiplication in the frequency domain. This relationship can be expressed as follows:

$$F[g * h] = F[g]F[h] \tag{7}$$

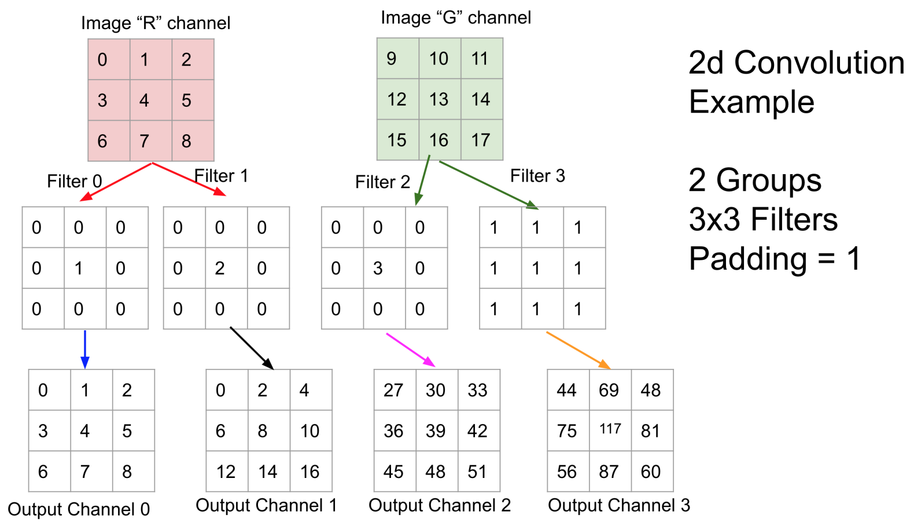


Figure 3: Visualization of a simple example using `groups=2`.

$$g * h = F^{-1}[F[g]F[h]] \tag{8}$$

To sharpen an image, the intuition is to emphasize the high-frequency components associated with edges and fine details. This can be achieved by convolving the image with a Laplacian kernel, which highlights regions of rapid intensity change.

$$I_{sharpen} = I + I * K \tag{9}$$

where I is original image, $*$ means convolution, K is laplacian kernel and $I_{sharpen}$ is sharpened image. In frequency domain, the above equation can be written as

$$I_{sharpen} = I + \text{IFFT}[\text{FFT}(I) \times \text{FFT}(K)] \tag{10}$$

Here, FFT represents Fast Fourier Transform and IFFT represents Inverse Fast Fourier Transform. Note, here the convolution from Equation before becomes multiplication of FFTs in frequency domain. You will implement `my_conv2d_freq()` in `part4.py` to convolve an image with the Laplacian filter in the frequency domain. Then, you will implement `my_sharpen_freq()` that takes the Laplacian kernel and sharpens the image by adding this convolved result back to the original image.

To perform a 2D Fourier transform and inverse 2D Fourier transform on an image, you should use the NumPy functions `numpy.fft.fft2()` and `numpy.fft.ifft2()`. You will also find `numpy.fft.ifftshift()` and `numpy.real()` very useful in this part. Note: Use a 3x3 Laplacian filter.

Instructions:
1. Implement the Convolution in Frequency Domain: Write the function `my_conv2d_freq()` that takes an image and a Laplacian kernel as input and returns the convolution result using the Fourier transform.
2. Implement Image Sharpening: Write the function `my_sharpen_freq()` that sharpens the image by adding the result of the convolution with the Laplacian kernel back to the original image.
3. Compare Results: Compare the original image with the sharpened image. Discuss the results, focusing on how sharpening in the frequency domain differs from spatial domain techniques.

5 Writeup

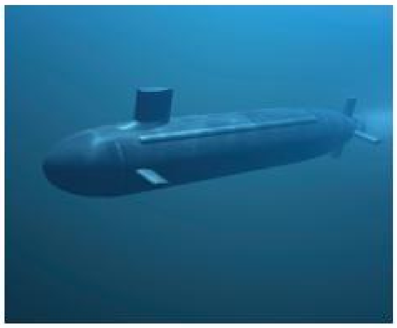
For this project (and all other projects), you must do a project report using the template slides provided to you. Do ***not*** change the order of the slides or remove any slides, as this will affect the grading process on Gradescope and you will be deducted points. In the report you will describe your algorithm and any decisions you made to write your algorithm a particular way. Then you will show and discuss the results of your algorithm. The template slides provide guidance for what you should include in your report. A good writeup doesn't just show results–it tries to draw some conclusions from the experiments. You must convert the slide deck into a PDF for your submission, and then assign each PDF page to the relevant question number on Gradescope.

If you choose to do anything extra, add slides *after the slides given in the template deck* to describe your implementation, results, and analysis. You will not receive full credit for your extra credit implementations if they are not described adequately in your writeup.

Data

We provide you with 5 pairs of aligned images which can be merged reasonably well into hybrid images. The alignment is super important because it affects the perceptual grouping (read the paper for details). We encourage you to create additional examples (e.g., change of expression, morph between different objects, change over time, etc.).

For the example shown in Figure 4, the two original images look like this:


(a) Submarine

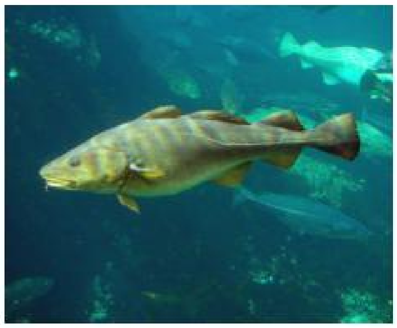

(b) Fish

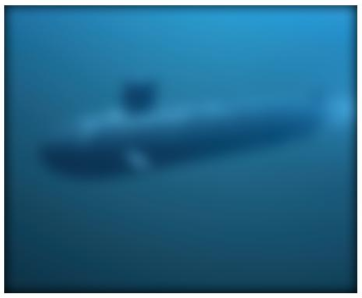
Figure 4

The low-pass (blurred) and high-pass version of these images look like this:
The high frequency image in Figure 5b is actually zero-mean with negative values, so it is visualized by adding 0.5. In the resulting visualization, bright values are positive and dark values are negative.

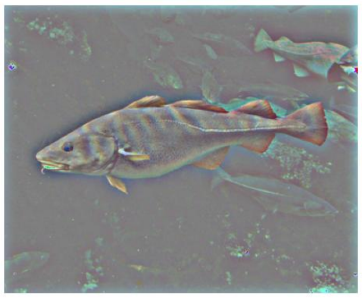
Adding the high and low frequencies together (Figures 5b and 5a, respectively) gives you the image in Figure 1. If you're having trouble seeing the multiple interpretations of the image, a useful way to visualize the effect is by progressively downsampling the hybrid image, as done in Figure 6. The starter code provides a function, `vis_image_scales_numpy()` in `utils.py`, which can be used to save and display such visualizations.

Potentially useful NumPy (Python library) functions

`np.pad()`, which does many kinds of image padding for you, `np.clip()`, which "clips" out any values in an array outside of a specified range, `np.sum()` and `np.multiply()`, which makes it efficient to do the convolution (dot product) between the filter and windows of the image. Documentation for NumPy can be found here or by Googling the function in question.



(a) Low frequencies of submarine image.



(b) High frequencies of fish image.

Figure 5

Forbidden functions

(You can use these for testing, but not in your final code). Anything that takes care of the filter operation or creates a 2D Gaussian kernel directly for you is forbidden. If it feels like you're sidestepping the work, then it's probably not allowed. Ask the TAs if you have any doubts.

Testing

We have provided a set of tests for you to evaluate your implementation. We have included tests inside `project-1.ipynb` so you can check your progress as you implement each section. When you're done with the entire project, you can call additional tests by running `pytest tests` inside the root directory of the project, as well as checking against the tests on Gradescope. *Your grade on the coding portion of the project will be further evaluated with a set of tests not provided to you.*

Bells & whistles (extra points)

For later projects there will be more concrete extra credit suggestions. It is possible to get extra credit for this project as well if you come up with some clever extensions which impress the TAs. If you choose to do extra credit, you should add slides ***at the end*** of your report further explaining your implementation, results, and analysis. You will not be awarded credit if this is missing from your submission.

Submission

This is very important as you will lose 5 points for every time you do not follow the instructions. You will submit two items to Gradescope:

1. `<your_gt_username>.zip` containing:
    - `src/`: directory containing all your code for this assignment
    - `cutoff_frequency.txt`: .txt file containing the best cutoff frequency values you found for each pair of images in data/
    - `setup.cfg`: setup file for environment, no need to change this file
    - `additional_data/`: (optional) if you use any data other than the images we provide, please include them here

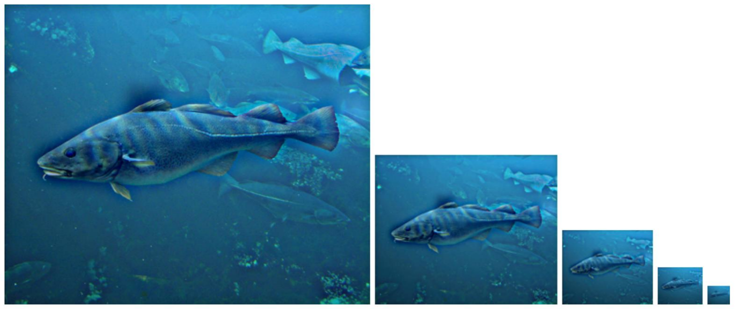


Figure 6

- `README.txt`: (optional) if you implement any new functions other than the ones we define in the skeleton code (e.g., any extra credit implementations), please describe what you did and how we can run the code. We will not award any extra credit if we can't run your code and verify the results.

2. `<your_gt_usernamme>_proj1.pdf` - your report

Credits

Assignment developed by Humphrey Shi, Aditya Kane, Dhruv Patel, Manushree Vasu, and Yue Zhao, based on a similar project by James Hays.