Assignment 3 - Report

EE5178: Modern Computer Vision

Vishnu Vinod

1 Introduction

In signal processing a filter is used to process a signal by filtering out components of the signal which are unnecessary or not required. In the given assignment we shall experimentally show that the convolution theorem holds in the case of 1D input. We shall also show the implementation of convolution the fourier space.

The second part of the assignment is based on creating hybrid images by virtue of low pass filtering and high pass filtering to give images which look a certain way when viewed closely and at a distance. These are discussed in relevant sections.

2 Filtering

In signal processing, filtering is a process of removing unwanted features or components from a signal. Most often, this means removing or suppressing some frequencies or frequency bands. Filtering operation can be linear, non-linear, space-variant or invariant. Let us first define and understand these terms.

A filter 'F' is called a linear filter if the output of the filter can be expressed as a linear combination of inputs in the signal space. It must satisfy the below given properties for two signals 'f' and 'g' and a scalar constant 'c':
• Additivity: $[F(f+g) = F(f) + F(g)]$
• Homogeneity: $[F(cf) = cF(f)]$

A filter 'F' is spatially invariant if any change δ in the input signal will directly translate as a change in the output signal. Mathematically we can write for a shift δ in the input signal the output signal shifts as: $y(m-\delta) = F(x(m-\delta))$.

Let us now analyse the filters given:

Filter 1: $y_k = x_{k+1} - x_k$

The above filter is both **Linear** and **Space-Invariant**. This can be seen very clearly from the fact that it is a simple linear combination of the signal space inputs and that a shift in the input signal will lead to identical shift in the output signal.

Thus it can be represented by an equivalent convolutional filter. This is given by:

$$kernel = \big[0, -1, 1\big]$$

We can see from the implementation of the filter that the convolved signal is the same as the computed signal.

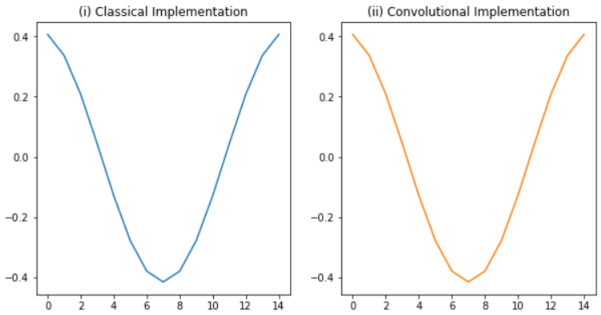

Figure 1: Filter 1 - Computational vs Convolutional Implementation

Filter 2: $y_k = x_k - \frac{1}{L+1}\sum_{i=0}^{L} x_i$

The above filter is both **Linear** and **Space-Invariant**. This can be seen very clearly from the fact that it is a simple linear combination of the signal space inputs and that a shift in the input signal will lead to identical shift in the output signal.

Thus it can be represented by an equivalent convolutional filter. Note that here $L = 15$. This is given by:

$$kernel = \frac{1}{16}\big[-1, ...15 \text{ times...}, 15, -1, ...15 \text{ times...}, -1\big]$$

We can see from the implementation of the filter that the convolved signal is the same as the computed signal.

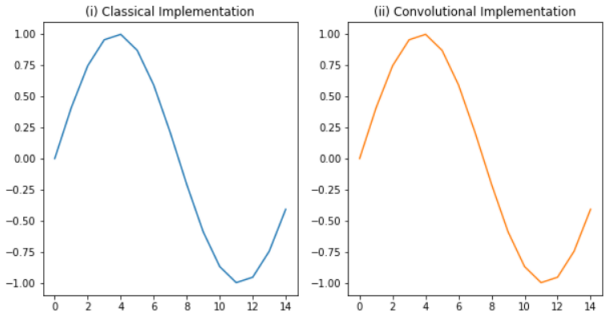

Figure 2: Filter 2 - Computational vs Convolutional Implementation

Filter 3: $y_k = median(\{x_i | i \in (l-2 : l+2)\})$

The above filter is **NOT Linear**. This can be seen very clearly from the fact that it is not a simple linear combination of the signal space inputs and the median operation carried out has no linear implementation.

Thus it cannot be represented by an equivalent convolutional filter. The output of the filter is given below:

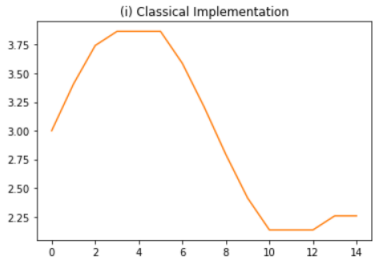

Figure 3: Filter 3 - Computational Implementation

Filter 4: $y_k = x_{k+0.5} - x_{k-0.5}$

Since we are supposed to use linear interpolation to get the values of $x_{k+0.5}$ and $x_{k-0.5}$ as follows:

$$x_{k+0.5} = (x_k + x_{k+1})/2$$

$$x_{k-0.5} = (x_k + x_{k-1})/2$$

Then we get that: $y_k = (x_{k+1} - x_{k-1})/2$.

The above filter is both **Linear** and **Space-Invariant**. This can be seen very clearly from the fact that it is a simple linear combination of the signal space inputs and that a shift in the input signal will lead to identical shift in the output signal.

Thus it can be represented by an equivalent convolutional filter. This is given by:

$$kernel = \frac{1}{2}[-1, 0, 1]$$

We can see from the implementation of the filter that the convolved signal is the same as the computed signal.

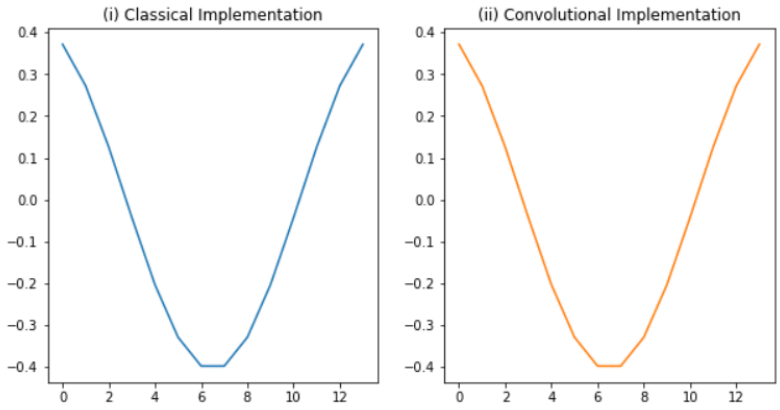

Figure 4: Filter 4 - Computational vs Convolutional Implementation

Filter 5: $y_k = |x_{k+0.5} - x_{k-0.5}|$

The above filter is **NOT Linear**. This can be seen very clearly from the fact that it is not a simple linear combination of the signal space inputs because the absolute value operation carried out has no linear implementation.

Thus it cannot be represented by an equivalent convolutional filter. The output of the filter is given below:

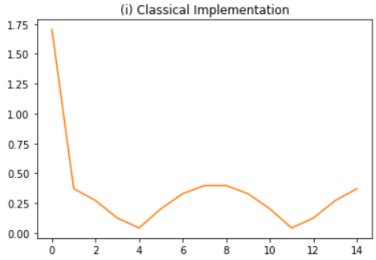

Figure 5: Filter 5 - Computational Implementation

Filter 6: $y_k = \frac{1}{5}\sum_{i=k-2}^{k+2} x_k$

The above filter is both **Linear** and **Space-Invariant**. This can be seen very clearly from the fact that it is a simple linear combination of the signal space inputs and that a shift in the input signal will lead to identical shift in the output signal.

Thus it can be represented by an equivalent convolutional filter. This is given by:

$$kernel = \frac{1}{5}[1, 1, 1, 1, 1]$$

We can see from the implementation of the filter that the convolved signal is the same as the computed signal.

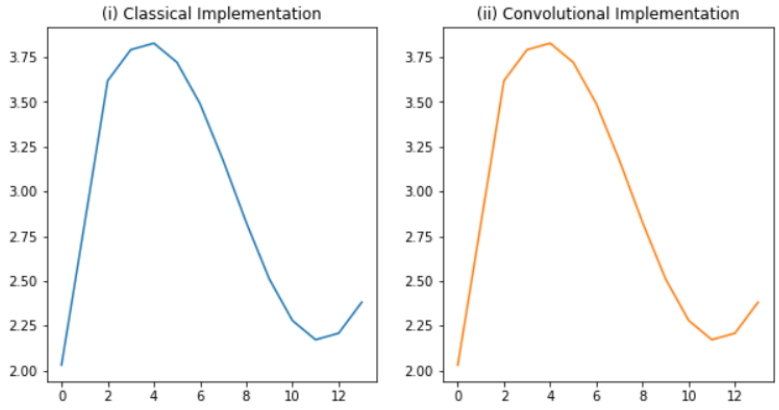

Figure 6: Filter 6 - Computational vs Convolutional Implementation

3 Fourier Space Filtering

We know from the one-dimensional convolution theorem that the convolving two functions in spatial coordinates (signal space) is equivalent to multiplying their fourier transformed versions in spectral coordinates (fourier space). In order to deal with the above discrete valued signal inputs we will use the **Discrete Fourier Transform** (DFT) and its inverse (IDFT).

The convolution theorem may be mathematically represented as follows:

$$DFT((f * g)) = DFT(f) \cdot DFT(g)$$

We can use this to convolve two functions very fast. The Discrete Fourier Transform itself is given by:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi kn}{N} i}$$

Furthermore the inverse DFT is given by:

$$X_k = \frac{1}{N} \sum_{n=0}^{N-1} x_n e^{\frac{2\pi kn}{N} i}$$

This can be easily implemented in python. An important point here is that when we are implementing DFT on a signal with a filter we will need to pad the input signal with zeros to ensure mathematical correctness. This can be easily

carried out. Then we get the following results for each of the four filters which are Linear and Spatially invariant.

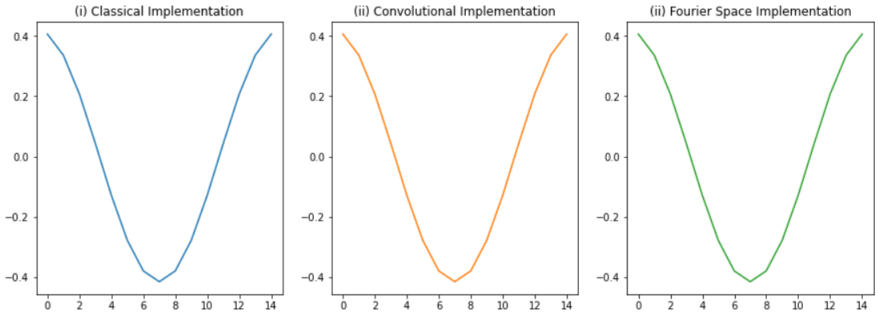

Figure 7: Fourier Implementation of Filter 1

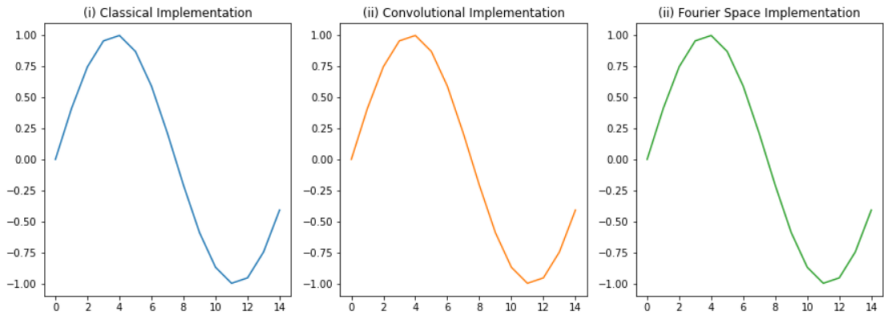

Figure 8: Fourier Implementation of Filter 2

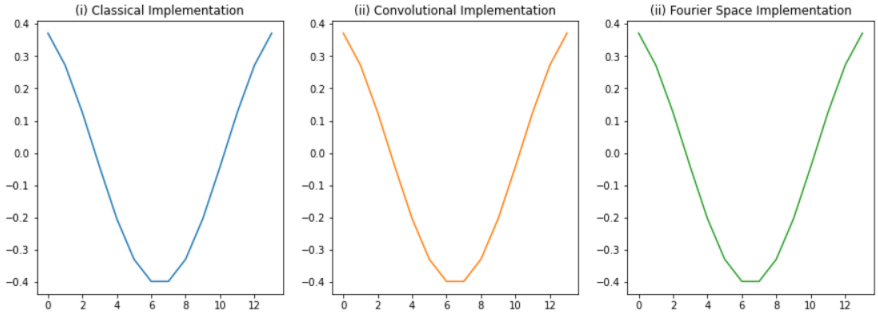

Figure 9: Fourier Implementation of Filter 4

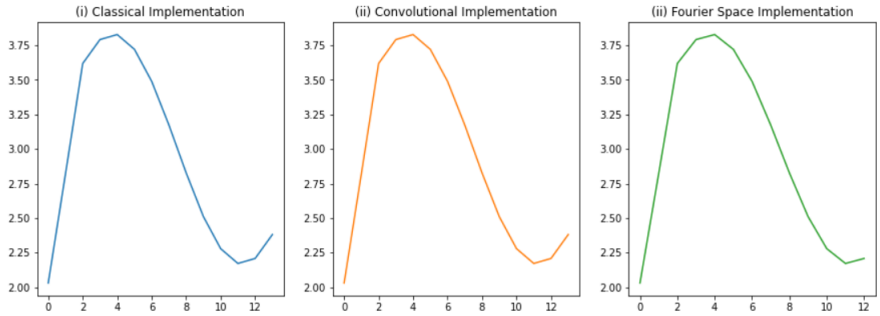

Figure 10: Fourier Implementation of Filter 6

A small note here is that the outputs obtained in the case of filter 2 and filter 4 were found to have been shifted slightly and we have clipped the ends of the outputs (only the last and/or the first value) to show only the parts of the function which are smooth. Furthermore the padding applied here is quintessential to getting the DFTs of the kernel and the signal which may be directly elementwise multiplied with each other.

4 Hybrid Images

The creation of hybrid images is a type of image merging which was a very old precursor of modern deep-learning based methods like NST (Neural Style Transfer). It basically works on the principle that the human eye when observing images from close will tend to focus on sharper details whereas when we view an image from afar we tend to average out nearby pixels to get "the bigger picture".

This information combined with the basics of frequency domain filtering allow us to merge two images such that one image is more pronounced when viewed from close by and the other is more pronounced when viewed from afar. Let us first understand the two types of filters to be used to achieve this:

- **Low Pass Filter:** A low pass filter allows only low frequency components to pass through. It carries out denoising and removes the high frequency components such as edges. It is used to smooth or blur the image. The gaussian filter used to carry out gaussian blurring is a type of low-pass filter.
  When viewing an image from close by the human eye carries out something akin to a low-pass filtering on the image allowing us to see larger overall patterns in the image rather than getting distracted by finer details.

- **High Pass Filter:** A low pass filter allows only low frequency components to pass through. It is used for edge detection and similar tasks where the fine details of the image are to be preserved. We can implement a high

pass filter by subtracting the output of a low-pass filter from the original image/signal.
When viewing the image from close by the human eye performs something akin to a high-pass filtering and allows us to notice and focus on the finer details of the image more than the larger overall patterns.

Further notes on the merging of the hybrid images:

- We carry out image pixel value clipping to ensure that the values stay in the range of 0 to 1. This is done trivially.
- The convolution function convolves the same filter across all channels to give the final image.
- The gaussian kernel is defined as follows for a standard deviation σ as having size $2ceil(3\sigma) + 1$. This ensures the kernel dimensions are odd. Here if $k = ceil(3\sigma)$ we have:

$$G_{ij} = \frac{1}{2\pi\sigma^2} \exp\Big(- \frac{(i-k)^2 + (j-k)^2}{2\sigma^2} \Big); 0 \leq i, j < 2k + 1$$

- If the image is grayscale, its shape will be of the form $[ih, iw]$. This must be changed into 3D format as $[ih, iw, ic = 1]$.
- For images with different sizes we need to ensure that the longer edges align together so that rescaling will lead to maximum overlap in the image sizes.
- If the images have different dimensions, we can rescale such that any one pair of dimensions match. Either width or height may be selected for this.
- Now for the case of the output image we merge the low-pass filtered part of the first image with the high-pass filtered second image such that the minimum of each dimension is taken. This is to ensure that we do not go out of bounds for any index in the image.
- We use the function defined in the *helpers.py* file to visualize the hybrid image at different scales.

Given below for each of the 7 pairs of images we have 2 hybrid images by considering the first image in the directory as image1 and image2 respectively. This is shown in the series of figures below. The values of sigma have been tuned visually for each hybrid image.

Figure 11: Bicycle and Motorcycle Hybrid Images

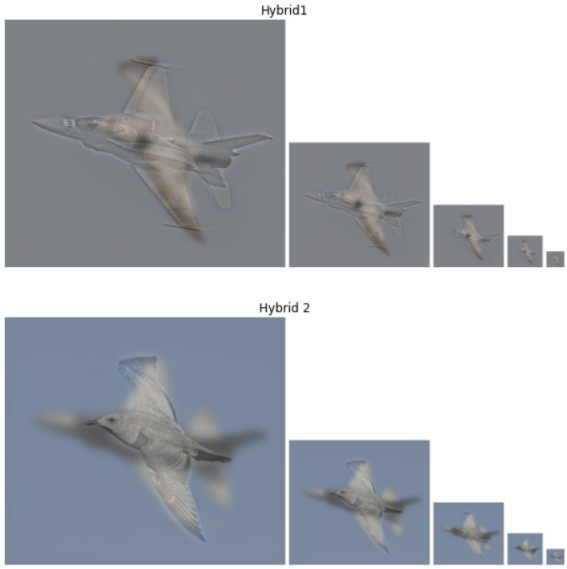

Figure 12: Bird and Plane Hybrid Images

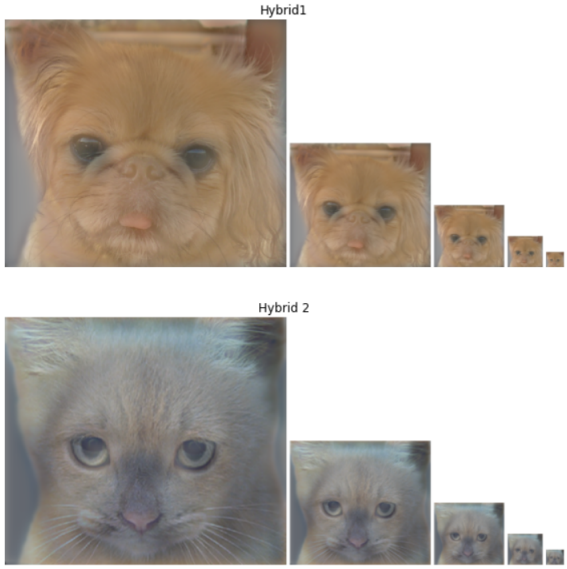

Figure 13: Cat and Dog Hybrid Images

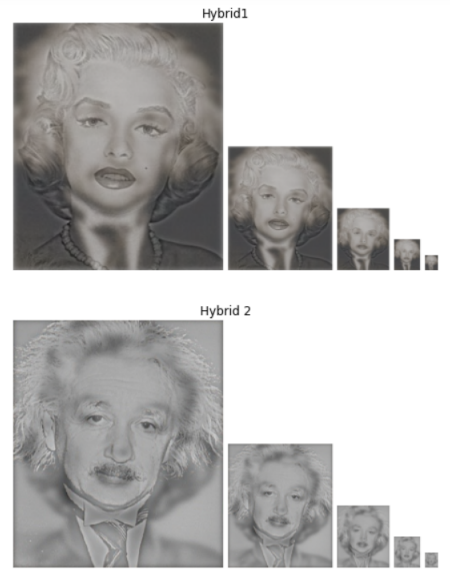

Figure 14: Albert Einstein and Marilyn Monroe Hybrid Images

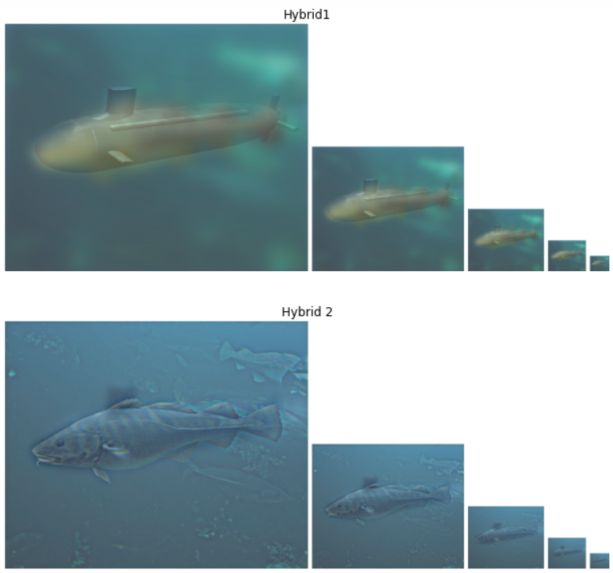

Figure 15: Fish and Submarine Hybrid Images

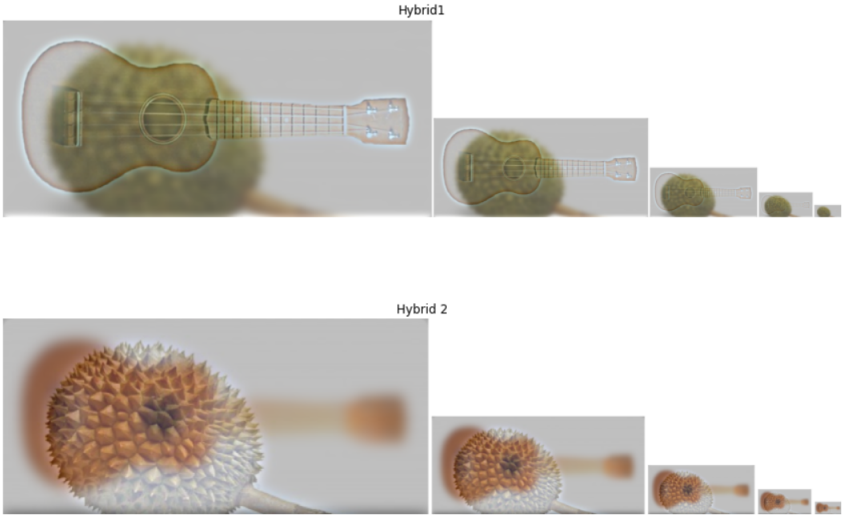

Figure 16: Durian and Ukulele Hybrid Images

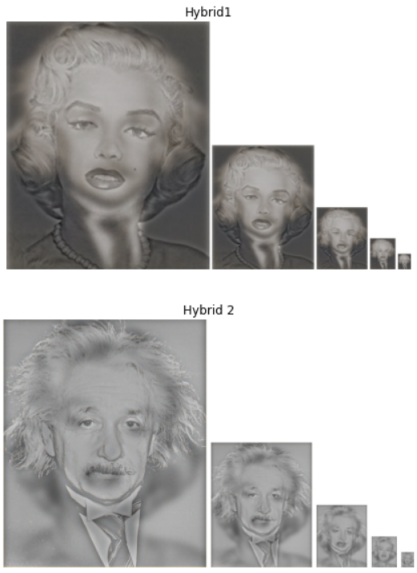

Figure 17: Albert Einstein and Marilyn Monroe Hybrid Images

Observations

Here we can see that the images have been successfully merged by taking the low-frequency components of one and merging it with the high frequency components of the other image. In the case of image set 6 and 7 the sizes of the images were different and thus they required rescaling and truncating before the output could be displayed.

It is important to note that when we visualize the image by repatedly downsampling it, the higher frequency information gets slowly masked out of the image. Thus at lower resolutions, the image becomes increasingly dominated by the lower frequency components of the image. This is what we observe as the disappearence of features from the image.