2D Transforms

Nina Tchirkova and Matt Brucker

January 9, 2018

Learning Goals

By the end of this overnight, you should be able to:

- Understand how a grayscale image can be represented as a 2-dimensional, discrete signal.
- Know how to take a 2-dimensional Discrete Fourier Transform of an image and what that means.
- Understand the idea of filtering in the frequency domain and how different frequencies map to different elements of the image.
- Apply filtering techniques (low-pass, high-pass, mean, derivative, etc.) to images in MATLAB.
- Understand the differences beteween DFTs and Discrete Cosine Transforms (DCTs).
- Apply DCTs for compression of image data.

Fourier Transforms in 2-Dimensions

Key Ideas

- The DFT can be taken for two dimensional signals
- The IDFT can also be taken of a two dimensional frequency spectrum.

Resources

- 2D DFT

As a review, the Discrete Fourier Transform transforms a finite length signal that starts as a function of time into a function of frequency. Just like any one dimensional signal (ex. audio) can be represented as a combination of one dimensional frequencies (see Figure 1), any two dimensional signals (ex. image) can be represented as a combination of two-dimensional frequencies (see Figure 2). A one-dimensional signal is just a list of values associated with specific moments in time. A two dimensional signal can be represented as a two dimensional matrix of values. That is why a gray scale image can be considered a two-dimensional signal - recall in the Eigenfaces

module that we represented gray scale images as two-dimensional matrices where each entry was a number 0-255 representing the brightness of a given pixel.

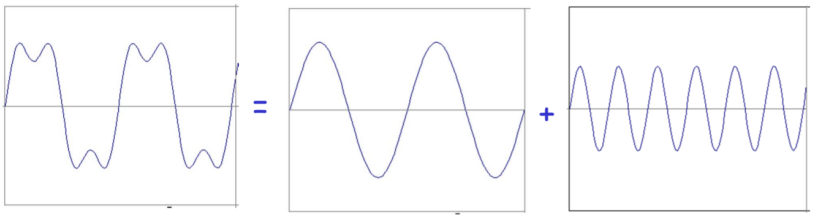

Figure 1: Example of combination of frequencies to create a wave.

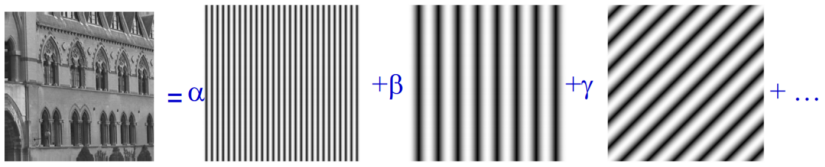

Figure 2: Example of two-dimensional frequencies combining to make an image

The definition for a two-dimensional discrete Fourier transform is thus very similar to a one dimensional discrete Fourier transform. These transforms are discrete because there is a finite number of values in the signal. The equation for a singular transform is below:

$$F(n) = \sum_{n=0} N - 1 f(k) e^{-j2\pi nk/N}$$

The definition of a two dimensional discrete Fourier transform is:

$$F(k,l) = \frac{1}{\sqrt{MN}} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f[m,n] e^{-j2\pi(\frac{k}{M}m + \frac{l}{N}n)}$$

In this formula, k and l are spatial frequencies and N and M are the dimensions of the original matrix. Even though the two-dimensional formula may seem intimidating, the process is not that much more complicated than completing a one dimensional transform. To take a two-dimensional Fourier transform, you take the one dimensional transform of each of the rows and then the one dimensional transform of each of the columns of the resulting matrix. However, doing this process manually would be quite cumbersome, thankfully MATLAB already has a function that does it for you called fft2.

As you may have expected, there is also an inverse two-dimensional Fourier transform that converts from the two-dimensional frequency domain back to the time domain. The equation for this is:

$$f(m,n) = \frac{1}{\sqrt{MN}} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} F[k,l] e^{j2\pi(\frac{k}{M}m + \frac{l}{N}n)} \quad (1)$$

MATLAB also has a function for this called ifft2.

Exercises

1. When you were introduced to the DFT, you learned how it can be represented as a matrix operation. Given a DFT matrix:

$$\mathbf{V} = \begin{bmatrix} \mathbf{v_1} \mathbf{v_2} \mathbf{v_3} \dots \mathbf{v_n} \end{bmatrix} \tag{2}$$

   Where each column is:

$$v_i = \frac{1}{\sqrt{N}} \begin{bmatrix} e^{\frac{j2\pi i}{N}} 0 \\ e^{\frac{j2\pi i}{N}} 2 \\ e^{\frac{j2\pi i}{N}} 2 \\ \vdots \\ e^{\frac{j2\pi i}{N}} (N-1) \end{bmatrix} \tag{3}$$

   How would you compute the DFT of a 2D signal as a matrix operation?

2. Download a black and white image from online, and take the 2D Fourier Transform of it. Helpful functions: `fft2`, `imread`, `mat2gray`, `fftshift`

3. Transform the Fourier transform of the image you downloaded using ifft in MATLAB. The image should look the same as the one you downloaded, but if you observe slight changes please record what you see. Helpful functions: `ifft2`, `ifftshift`

Applying Filters in 2 Dimensions

Key Ideas

- Filters that are applied to one dimensional signals can also be applied to two dimensional signals
- Image editing techniques often use different types of filters in the frequency space
- Applying a filter in the frequency space is the same as convolving a signal in the time domain with that filter

Resources

- MATLAB
- MATLAB

As the functions and properties of one-dimensional and two dimensional Fourier transforms have carried, there are also many possibilities for filtering two-dimensional Fourier transforms. A low pass filter applied to the transform of an image would cause higher frequency properties (such as noise) to be eliminated while a high pass filter would take a way a lot of the details of the image. See Figure 2 for a visual representation of this concept.

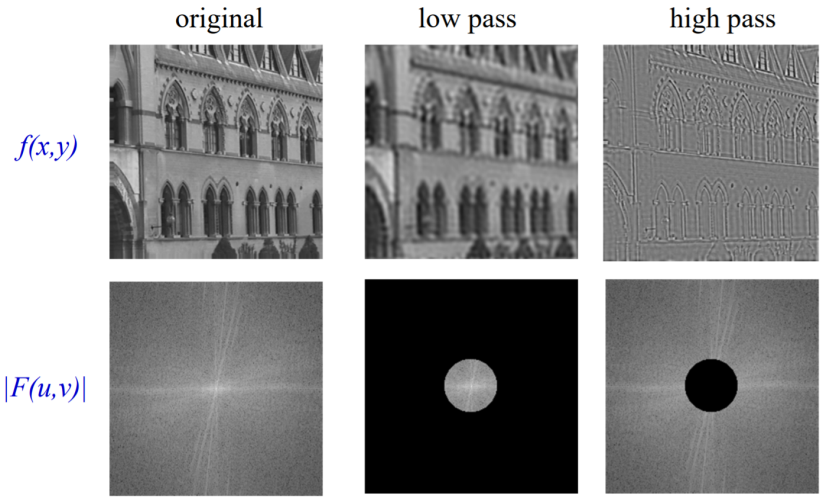

Figure 3: Visual representations of how low pass and high pass filters affect an image and its Fourier transform.

Other types of filters cause other types of effects on images. These filters are often used in image processing operations. In order to implement a filter on a Fourier transform it is helpful to remember the following theorem:

$$f(x,y) * h(x,y) = F(u,v)H(u,v)$$

If H(u,v) represents a filter in the frequency domain then, multiplying the Fourier transform of a signal with a filter is the same as convolving the original image with the filter in the spatial domain. Since it is much easier to define a filter in the frequency domain and multiplication is an easier operation than convolution, bringing an image into the frequency domain in order to filter it makes sense.

Exercises

1. Using the MATLAB script `filtertesting.m`, apply a low pass filter to an image and observe the results. Try changing the R value and see how this effects the filter and the image.
2. Apply a high pass filter to an image and observe the results. Try changing the R value and see how this effects the filter and the image.

3. Research a different type of filter and implement it to apply it to the image. Are the results the same as you expected? For inspiration, you can refer to the `fspecial` function in MATLAB.

   filtertesting.m

Transforming for Image Compression: The Discrete Cosine Transform

Key Ideas

- The DCT is a Fourier-like transform based on cosine waves that can represent a discrete signal as a linear combination of different-frequency cosine waves.
- The DCT has properties (boundary conditions, energy compaction, etc.) that make it ideal for compressing image-like information
- The DCT can applied to 2D signals in the same way as the DFT.
- JPEG compression uses the DCT to compress image information and reduce high frequencies that the human eye doesn't notice.

Helpful Resources

- The Discrete Cosine Transform - a good overview of how the DCT works.
- MIT DCT Paper - a more in-depth look at the DCT.

In the Eigenfaces module, you were introduced to the idea of compression - essentially getting rid of some of the less noticeable components of the data in order to reduce the amount of storage needed for the data. You found that, by reconstructing each face from a subset of the eigenfaces, you could get a reconstruction that looked fairly close to the original image while also reducing the dimensionality - and thus the amount of data to store. In this section, we'll be exploring a new transform - the **Discrete Cosine Transform** - that has properties which make it well-suited for compressing images.

Discrete Cosine Transforms

The Discrete Cosine Transform is a transform in the Fourier family of transforms. That is, it involves the projection of a signal onto a number of sinusoidal basis vectors - except now the basis vectors are

cosine waves instead of complex exponentials as in the DFT. We can define a single basis vector of the DCT matrix as:

$$v_i = \alpha_i \begin{bmatrix} cos(\frac{\pi}{2N}i) \\ cos(\frac{\pi}{N}(1+\frac{1}{2})i) \\ cos(\frac{\pi}{N}(2+\frac{1}{2})i) \\ \vdots \\ cos(\frac{\pi}{N}((N-1)+\frac{1}{2})i) \end{bmatrix} \tag{4}$$

where N is the size of the image. The DCT matrix then becomes:

$$\mathbf{V} = \begin{bmatrix} \mathbf{v_1 v_2 v_3 \dots v_n} \end{bmatrix} \tag{5}$$

Note that α_i is a coefficient equal to:

$$\alpha_i = \begin{cases} \frac{1}{\sqrt{N}} & i = 0 \\ \sqrt{\frac{2}{N}} & 1 \leq i \leq N-1 \end{cases} \tag{6}$$

The purpose of α_i is to make the DCT matrix orthonormal; this way, we can transpose the DCT matrix to find its inverse.

Properties of the DCT

The DCT has a number of properties that differentiate it from the DFT. The first property should be fairly noticeable: as it's only using a real function (the cosine), the output of the DCT is always real. Other properties of the DCT have important implications for how and where it's used.

Exercises

1. First, let's think about what the inverse DCT looks like. Based on how the DCT is defined for 2D signals, how would you compute the inverse DCT of a 2D signal? *Hint: remember that the DCT matrix is orthonormal.*

2. Recall from when the DFT was introduced that high frequencies and low negative frequencies are the same, due to the periodic nature of the DFT. Based on the formula for a DCT basis vector at a certain frequency:

$$cos(\frac{\pi}{N}(k+\frac{1}{2})i) \tag{7}$$

   where i is the frequency, determine whether the DCT exhibits any periodicity. *Hint: try different multiples of N as the frequency.*

3. There are two main properties of the DCT, based on properties of real-world signals, which make it advantageous for applications to image data. Real-world signals are almost always `pink` in nature, which means that the power of a signal is proportional to

$\frac{1}{f}$. The DCT has high energy compaction, which means that low-frequency content of a signal is highly concentrated together. This is useful for the DCT, as it means that most of the power of the signal will be concentrated in the first few terms of the DCT.

(a) Load the Handel default sound file in MATLAB. Take the DCT and DFT of it, and try reconstructing the original signal from the first 30,000, 20,000, and 10,000 elements of each. How is the signal affected by the DCT vs. the DFT?

4. Another important property of DCTs - and their relation to DFTs - is their *edge behavior*. Because these transforms turn signals into sums of exponentials, which are defined for the set of real numbers, these transforms imply an extension of the original signal infinitely in the time domain. However, different transforms handle this extension differently; the DFT simply repeats the original signal infinitely, whereas the DCT is even at boundaries, meaning that anything past the boundaries of the signal is just a reflection of the signal (there are other types of DCTs that are odd, but we focus on a DCT that is even). Note that in the DFT, a signal

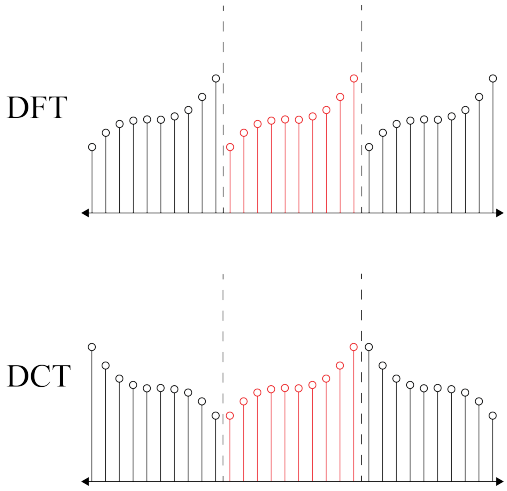

Figure 4: A representation of how each transform implicitly extends a signal in the time domain. The sampled signal is shown in red; each respective transform's extension of it is to the right and left.

with a very different starting and ending value has a large jump - or *boundary discontinuity* - between the original signal and its extensions. This results in a lot of high frequencies showing up to compensate. However, in the DCT, there is no discontinuity at boundaries, and thus fewer high frequencies showing up, making it more advantageous for transforming image data, which may have very different edges.

5. Another way to look at this property is to look at how various signals project onto the basis vectors of each transform. Create a

DFT matrix of a reasonable size (say, 64x64) using `dftmtx`, and a DCT matrix using `dctmtx`. Plot the first few columns of the DFT and the DCT. how are the beginning and end points different between the DCT and DFT?

6. If we have a series of 50 data points that lie on a straight line, how would the reconstruction of the signal from the first 15 values of the DFT differ from the reconstruction based on 15 elements of the DCT? Use the `:` operator in MATLAB to create a series of data points, and test your hypothesis by reconstructing the original data from the first few basis vectors of the transform.

2D DCTs and JPEG Compression

Figure 5: An image of a flower before and after performing JPEG compression.

Now that we understand why the DCT is a useful transform to apply to image-like signals, we can start applying them to 2D signals. Remember from the previous section that the row-wise and column-wise transforms are VX and XV^T, respectively. Thus, the 2D DCT is formally defined as:

$$X_{i,j} = \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} x_{n,m} cos(\frac{\pi}{N}(n+\frac{1}{2})i) cos(\frac{\pi}{M}(m+\frac{1}{2})j) \tag{8}$$

and can be computed by taking VXV^T, where V is the DCT matrix. This results in an NxN matrix where the zero frequency is in the top-left corner and frequency increases as you move down and to the right.

Exercises

Helpful Links:

- JPEG Pt. 1 (Computerphile) https://www.youtube.com/watch?v=n_uNPbdenRs

- JPEG Pt. 2 (Computerphile) https://www.youtube.com/watch?time_continue=769&v=Q2aEzeMDHMA

One very common use of the DCT is in performing JPEG compression. As you found in the earlier sections, much of the information in real-world signals is concentrated in a few lower frequencies, and the DCT does a very good job of compressing information into a few low frequency elements. The basic idea behind JPEG compression is to take the DCT of an image, reduce most of the high-frequency components to the extent that they can be rounded to 0, and encode the DCT using a method that takes advantage of repeated values to reduce storage size even more.

1. The first step in JPEG compression is to do some pre-processing. To start, we need to take an RGB image and convert it to a different color space. Load an image in MATLAB with `imread('test.jpg')`. You should get an NxNx3 matrix, with each NxN slice of it representing a color channel.
   Colorspaces
   In JPEG compression, the colorspace used is YCbCr. YCbCr is comprised of a luminance (Y) component which controls the brightness, and chrominance (Cb and Cr) components that control the color aspects. This colorspace is particularly useful for JPEG compression because the human eye doesn't distinguish color elements as easily as brightness, so by separating the two, we can greatly reduce color without it being very noticeable. To convert an RGB pixel to YCbCr, we can use matrix operations as so:

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.334 & 0.500 \\ 0.500 & -0.419 & -0.081 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix} \tag{9}$$

   (a) Write code in MATLAB to input an NxNx3 RGB matrix and return an NxNx3 YCbCr matrix.

2. An image also has to be centered around 0 in order to take its DCT. In a standard 8-bit image, pixel values fall in the range [0,255]. To center the matrix, we have to subtract 128 from each element.

3. Next, the image is broken up into 8x8 (or smaller) chunks. To understand why this is done, let's think about the number of operations needed to compute the DCT.
   A Note on Computational Complexity
   In computer science, the *computational complexity* of a function or algorithm is, in its simplest terms, the number of operations

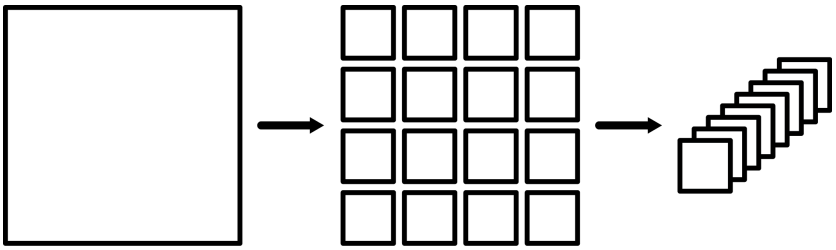

Figure 6: An illustration of how an image is broken up into 8x8 blocks and stacked up in JPEG compression.

needed to compute the function on an input of an arbitrary size. We can look at the computational complexity of the DCT to understand why it's advantageous to break an image into smaller chunks.

(a) Consider the case of taking the DCT of a 32x32 image. If we consider the multiplication of two elements in matrices to be a single operation, how many operations would it take to compute a dot product of a row and column vector? How many operations would it take to compute the entire DCT? (Don't forget that you have to take both the row and column DCTs)

(b) Now consider the situation where the image is broken into a 4x4 grid of 8x8 pixel chunks. Find the number of computations required to take the DCT of each chunk, and multiply that number by 16 to find the total number of operations to take the DCT in this case. Is it less computationally expensive to compute the DCT of everything, or of the individual chunks?

The file create_blocks.m in the provided resources handles the process of dividing a single image into chunks.

4. Now, we're ready to take the DCT of each block. In MATLAB, take the DCT of each block using `dct2`.

5. Next, we use quantization to get rid of information in certain frequencies. The objective of quantization is to make as many frequencies the same as possible, usually by driving them to zero. This is accomplished by dividing each DCT chunk by a *quantization matrix* and rounding values down. Quantization is formally defined as:

$$F_{quant} = Round(\frac{F(u,v)}{Q(u,v)}) \tag{10}$$

where Q(u,v) is the quantization matrix corresponding to a given color channel. Note that the dividing operation between these two matrices is elementwise division. For YCbCr, the quantization

matrices are:

$$Q_Y = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix} \tag{11}$$

$$Q_C = \begin{bmatrix} 17 & 18 & 24 & 47 & 99 & 99 & 99 & 99 \\ 18 & 21 & 26 & 66 & 99 & 99 & 99 & 99 \\ 24 & 26 & 56 & 99 & 99 & 99 & 99 & 99 \\ 47 & 66 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \end{bmatrix} \tag{12}$$

Recall from earlier that JPEG uses YCbCr colorspace because it separates brightness and color components, and brightness changes are more easily seen by the human eye than color. This manifests itself in the quantization matrices: the numbers increase at much lower frequencies in the color-channel matrix than in the brightness matrix; thus, certain lower frequencies are reduced more in the color channels than in the brightness channel.

Write a function in MATLAB to compute the quantized version of an 8x8 DCT matrix.

6. The final step in compressing the image is to encode the information. Huffman coding is a commonly-used method of encoding JPEG information - this is because it takes advantage of repeated elements in whatever information you want to encode, and the entire purpose of quantization is to get as many repeated elements as possible. Huffman coding depends on two functions: creating the Huffman dictionary, and performing the actual encoding based on the dictionary. You can use get_dicts.m and encode.m to perform the dictionary creation and encoding, respectively.

7. Now, we have a fully compressed an encoded JPEG - this is what would actually be stored on your computer, since it's the most compact way to store the information. Look at what `encode.m` outputs. If each element of the output is one bit, how does the size of the compressed information compare to the original information? *(Hint: each R/G/B pixel is 8 bits normally.)*

8. Now that the image has been encoded, we can decode it to see what the image looks like once it's been compressed. Based on the code to encode an image, write the code to decode an image by performing the inverse operations in backward order. Note that the step of *dequantization,* where the DCT is multiplied by the quantization matrix to undo the quantization, is where the information loss actually occurs; everything that was rounded off in the quantization step will remain rounded, and thus information is list. Compare the original image with the JPEG reconstruction. How do the two images look different? How could we reduce or improve the quality of the image?