FOURIER TRANSFORM IN IMAGE PROCESSING

Jean Baptiste Joseph Fourier

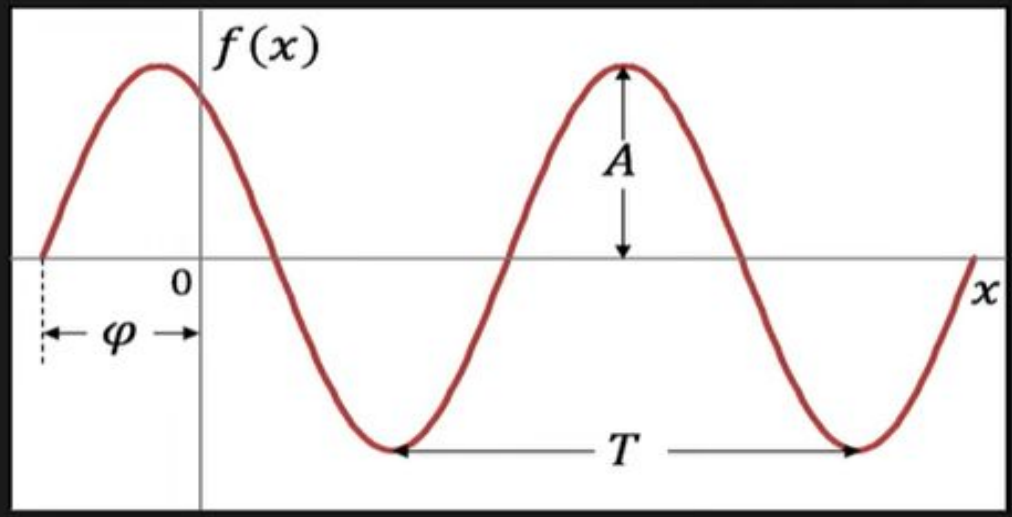
(1768-1830)

Any Periodic Function can be rewritten as a Weighted Su

of Infinite Sinusoids of Different Frequencies.

Sinusoid

$$f(x) = A\sin(2\pi ux + \varphi)$$



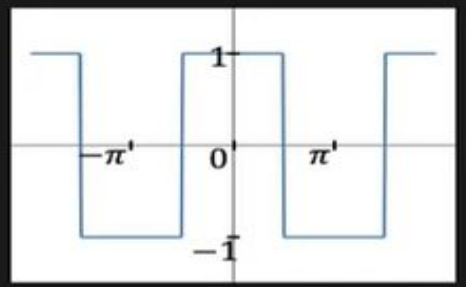
A: Amplitude

T: Period

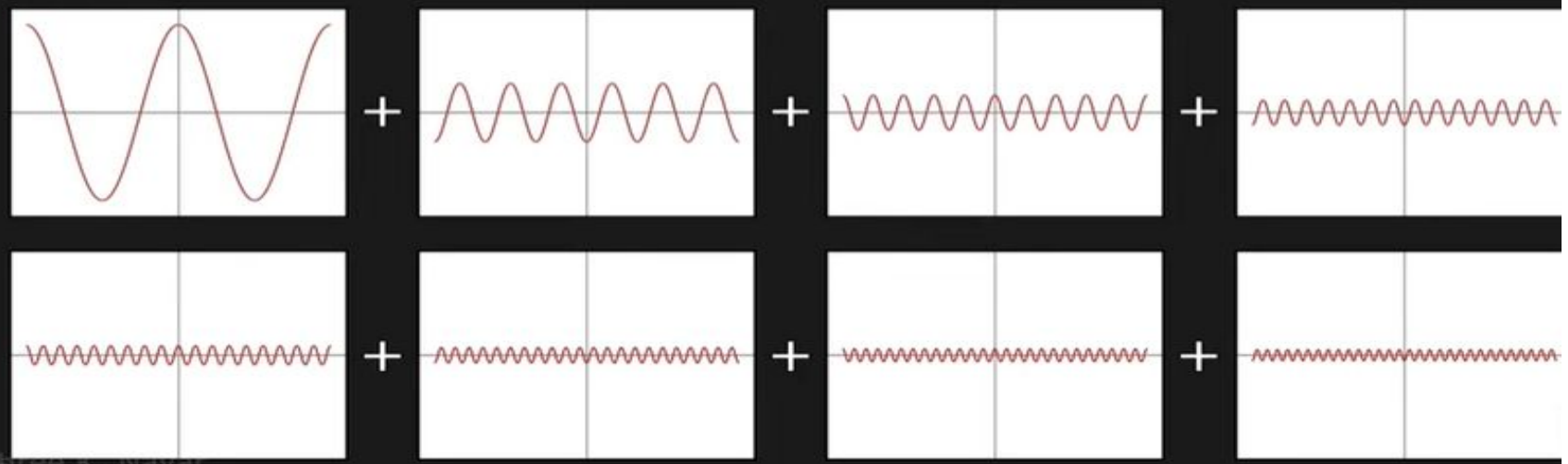
φ: Phase

u: Frequency $(1/T)$

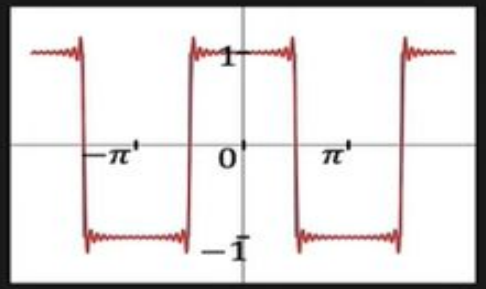
Fourier Series



Square Wave
(Period 2π)

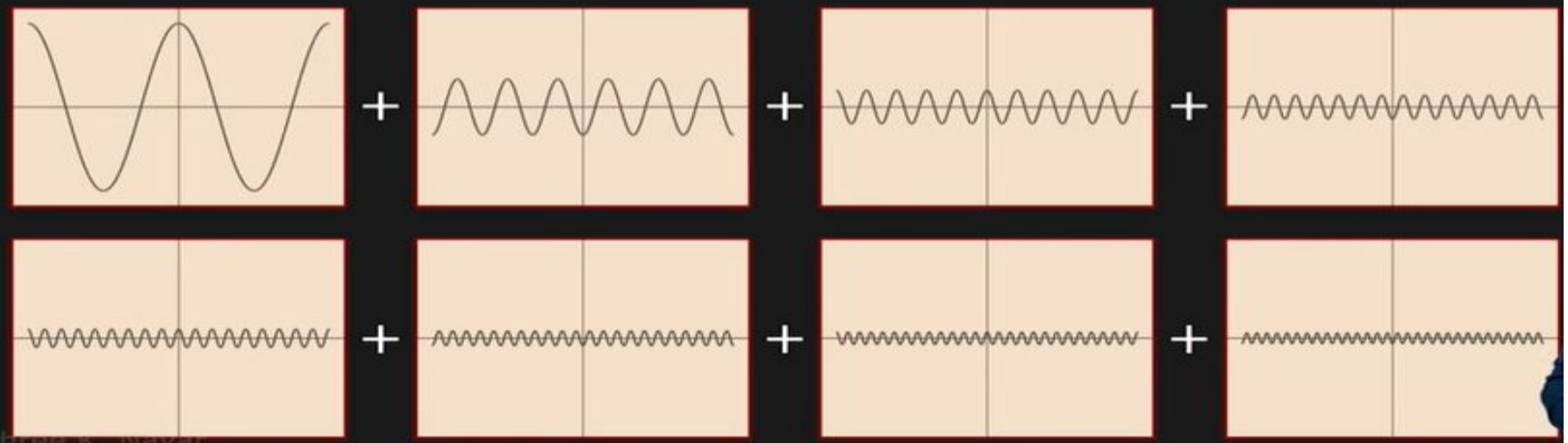


Fourier Series

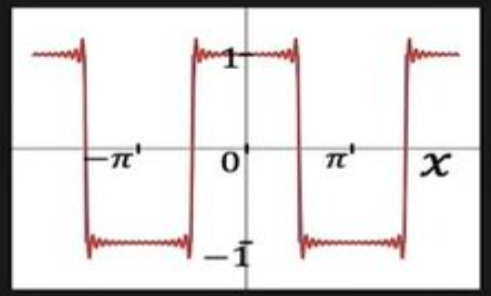


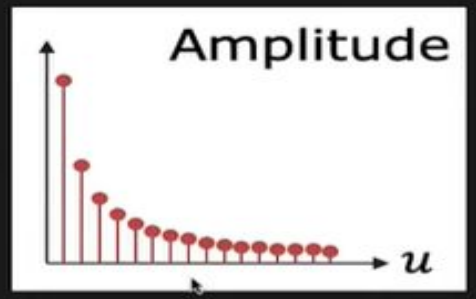
Square Wave (Period 2π)

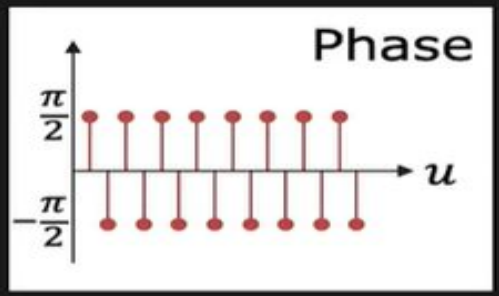
Sum of Sinusoid

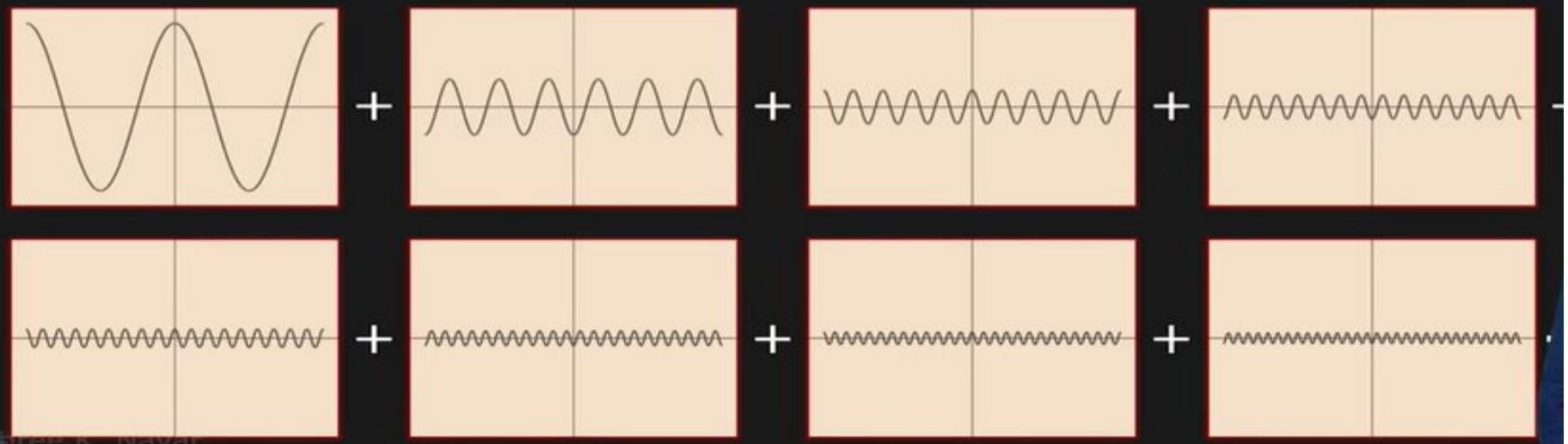


Frequency Representation of Signal

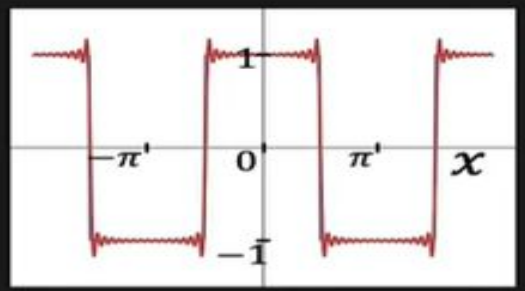


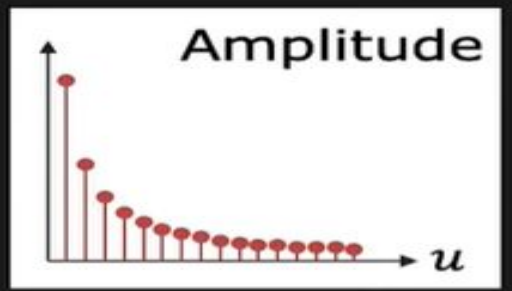


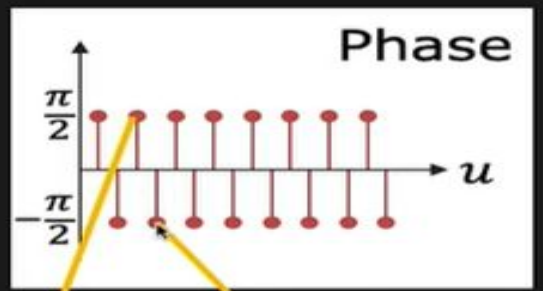


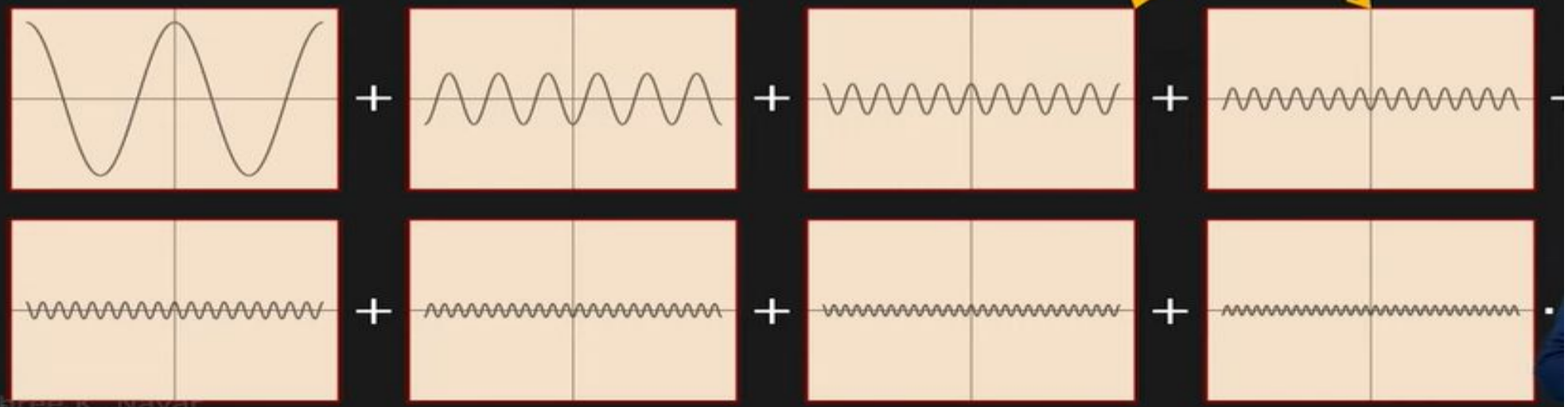


Frequency Representation of Signal

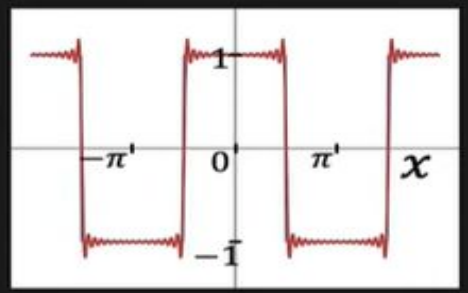


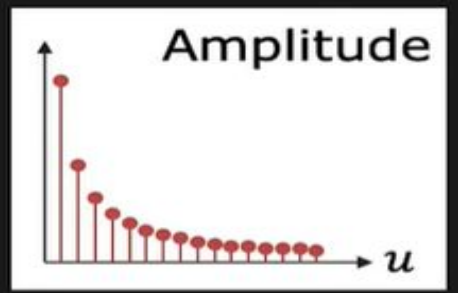


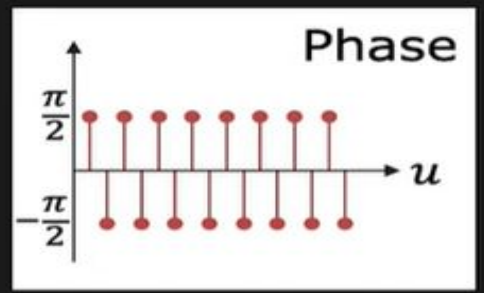




Fourier Transform (FT)



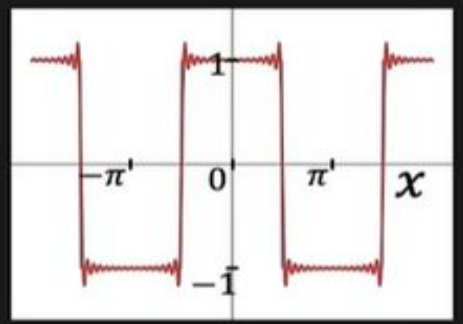


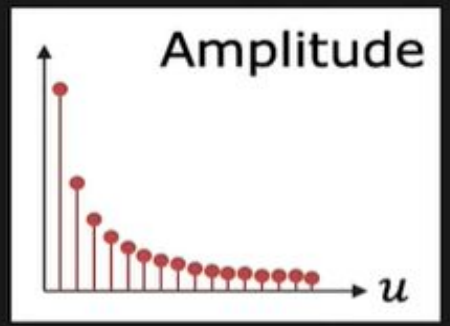


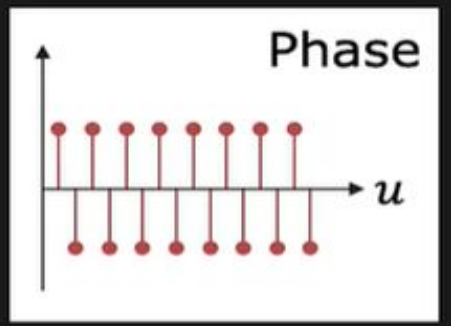
Represents a signal $f(x)$ in terms of Amplitudes and Phases of its Constituent Sinusoids.

$f(x)$ → FT → $F(u)$

Inverse Fourier Transform (IFT)







Computes the signal $f(x)$ from the Amplitudes and Phases of its Constituent Sinusoids.

$f(x)$ ← IFT ← $F(u)$

Finding FT and IFT

Fourier Transform:

$$F(u) = \int_{-\infty}^{\infty} f(x) e^{-i2\pi ux} dx$$

x: space

u: frequency

$e^{i\theta} = \cos\theta + i\sin\theta$

$i = \sqrt{-1}$

Inverse Fourier Transform:

$$f(x) = \int_{-\infty}^{\infty} F(u) e^{i2\pi ux} du$$

Complex Exponential (Euler Formula)

$$\boxed{e^{i\theta} = \cos\theta + i\sin\theta} \qquad i = \sqrt{-1}$$

Expand $e^{i\theta}$ using Taylor Series:

$$e^{i\theta} = 1 + i\theta + \frac{(i\theta)^2}{2!} + \frac{(i\theta)^3}{3!} + \frac{(i\theta)^4}{4!} + \frac{(i\theta)^5}{5!} + \frac{(i\theta)^6}{6!} + \dots$$

$$e^{i\theta} = \left(1 - \frac{\theta^2}{2!} + \frac{\theta^4}{4!} - \frac{\theta^6}{6!} + \cdots\right) + i\left(\theta - \frac{\theta^3}{3!} + \frac{\theta^5}{5!} - \frac{\theta^7}{7!} + \cdots\right)$$

Complex Exponential (Euler Formula)

$$\boxed{e^{i\theta} = \cos\theta + i\sin\theta} \qquad i = \sqrt{-1}$$

Expand $e^{i\theta}$ using Taylor Series:

$$e^{i\theta} = 1 + i\theta + \frac{(i\theta)^2}{2!} + \frac{(i\theta)^3}{3!} + \frac{(i\theta)^4}{4!} + \frac{(i\theta)^5}{5!} + \frac{(i\theta)^6}{6!} + \dots$$

$$e^{i\theta} = \underbrace{\left(1 - \frac{\theta^2}{2!} + \frac{\theta^4}{4!} - \frac{\theta^6}{6!} + \cdots\right)}_{\cos\theta} + i\underbrace{\left(\theta - \frac{\theta^3}{3!} + \frac{\theta^5}{5!} - \frac{\theta^7}{7!} + \cdots\right)}_{\sin\theta}$$

Fourier Transform is Complex!

$F(u)$ holds the Amplitude and Phase of the sinusoid of frequency u.

$$F(u) = \int_{-\infty}^{\infty} f(x) e^{-i2\pi ux} dx$$

$$F(u) = \mathfrak{Re}\{F(u)\} + i\,\mathfrak{Im}\{F(u)\}$$

Fourier Transform is Complex!

$F(u)$ holds the Amplitude and Phase of the sinusoid of frequency u.

$$F(u) = \int_{-\infty}^{\infty} f(x) e^{-i2\pi ux} dx$$

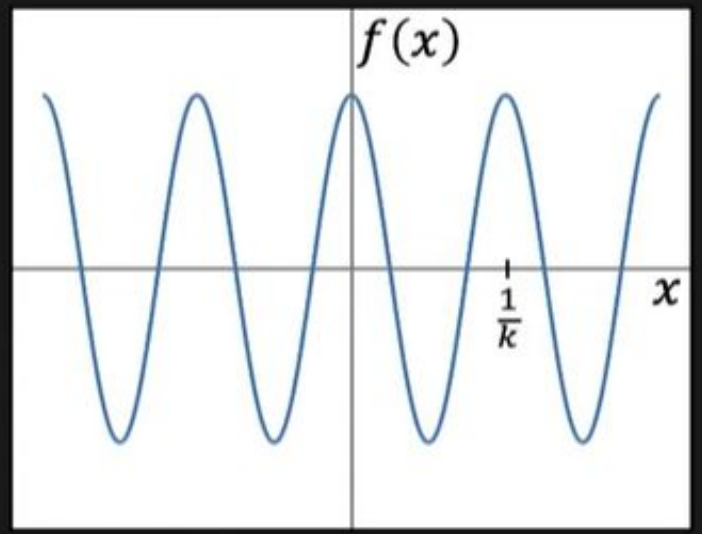
$$F(u) = \Re\mathrm{e}\{F(u)\} + i\,\Im\mathrm{m}\{F(u)\}$$

Amplitude: $A(u) = \sqrt{\Re\mathrm{e}\{F(u)\}^2 + \Im\mathrm{m}\{F(u)\}^2}$

Phase: $\varphi(u) = \mathrm{atan2}(\Im\mathrm{m}\{F(u)\}, \Re\mathrm{e}\{F(u)\})$

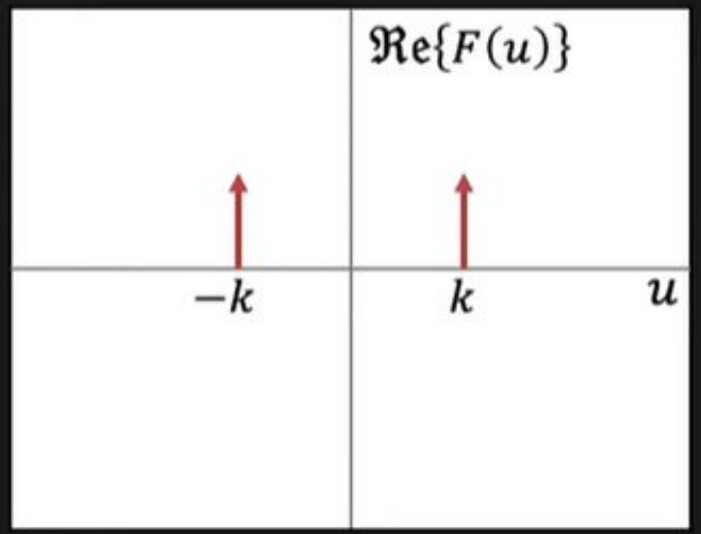
Fourier Transform Examples

Signal $f(x)$



$$f(x) = \cos 2\pi kx$$

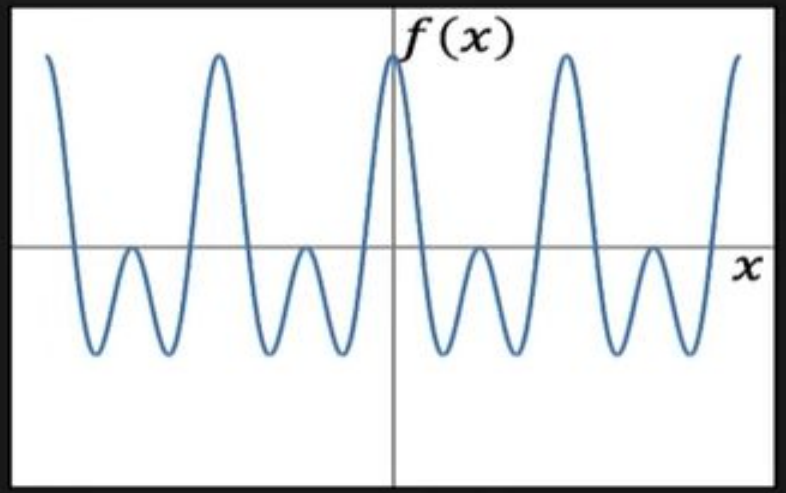
Fourier Transform $F(u)$



$$F(u) = \frac{1}{2}[\delta(u+k) + \delta(u-k)]$$

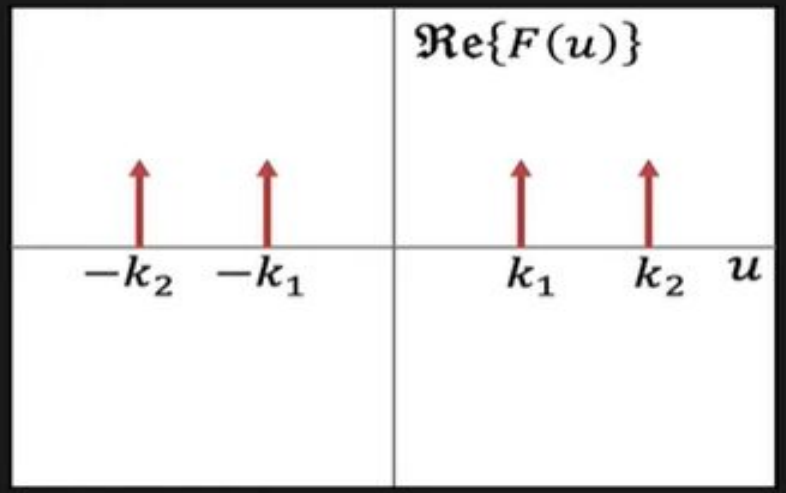
Fourier Transform Examples

Signal $f(x)$



$$f(x) = \cos 2\pi k_1 x + \cos 2\pi k_2 x$$

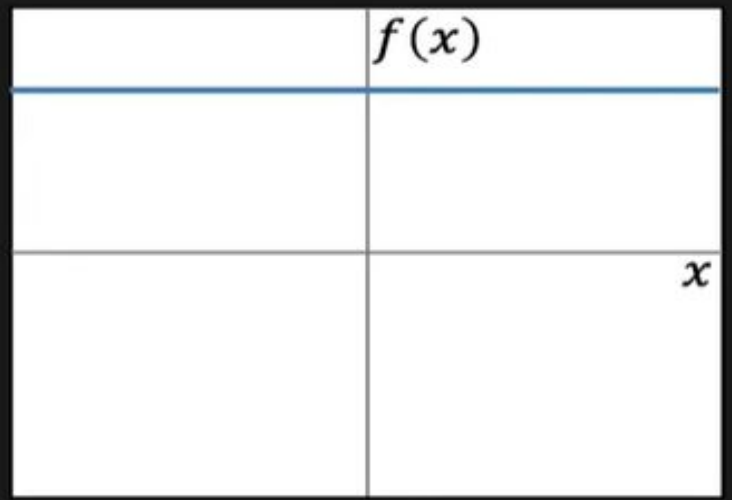
Fourier Transform $F(u)$



$$F(u) = \frac{1}{2}\left[\delta(u + k_1) + \delta(u - k_1) + \delta(u + k_2) + \delta(u - k_2)\right.$$

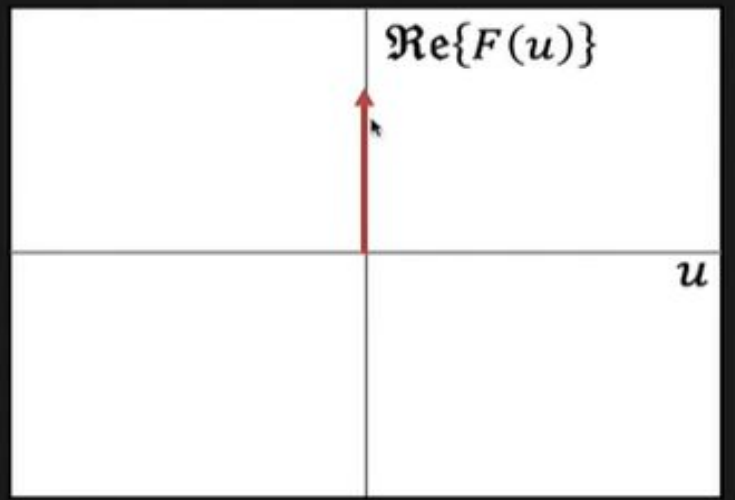
Fourier Transform Examples

Signal $f(x)$



$$f(x) = 1$$

Fourier Transform $F(u)$



$$F(u) = \delta(u)$$

Properties of Fourier Transform

Property	Spatial Domain	Frequency Domain
Linearity	$\alpha f_1(x) + \beta f_2(x)$	$\alpha F_1(u) + \beta F_2(u)$
Scaling	$f(ax)$	$\frac{1}{\lvert a \rvert} F\left(\frac{u}{a}\right)$
Shifting	$f(x-a)$	$e^{-i2\pi ua} F(u)$
Differentiation	$\frac{d^n}{dx^n}(f(x))$	$(i2\pi u)^n F(u)$

Applications of Fourier Transform in Image Processing

Image Filtering: The Fourier Transform enables the application of frequency domain filters to images. By converting an image to the frequency domain using the Fourier Transform, various types of filters can be applied, such as low-pass, high-pass, and band-pass filters. Filtering in the frequency domain is often more efficient than filtering in the spatial domain.

Image Compression: The Fourier Transform is the basis for many image compression techniques. One popular method, called the Discrete Cosine Transform (DCT), is a variant of the Fourier Transform that is widely used in image compression algorithms such as JPEG. The DCT converts an image into a set of frequency coefficients that can be quantized and encoded more efficiently.

Image Reconstruction: The Fourier Transform can be used to reconstruct an image from its frequency components. By taking the inverse Fourier Transform of a modified frequency domain representation of an image, it is possible to remove noise, artifacts, or unwanted elements from the image.

Image Registration: Image registration is the process of aligning and overlaying two or more images of the same scene taken from different perspectives or at different times. The Fourier Transform can be used to calculate the cross-correlation between images in the frequency domain, which helps in determining the spatial alignment between images.

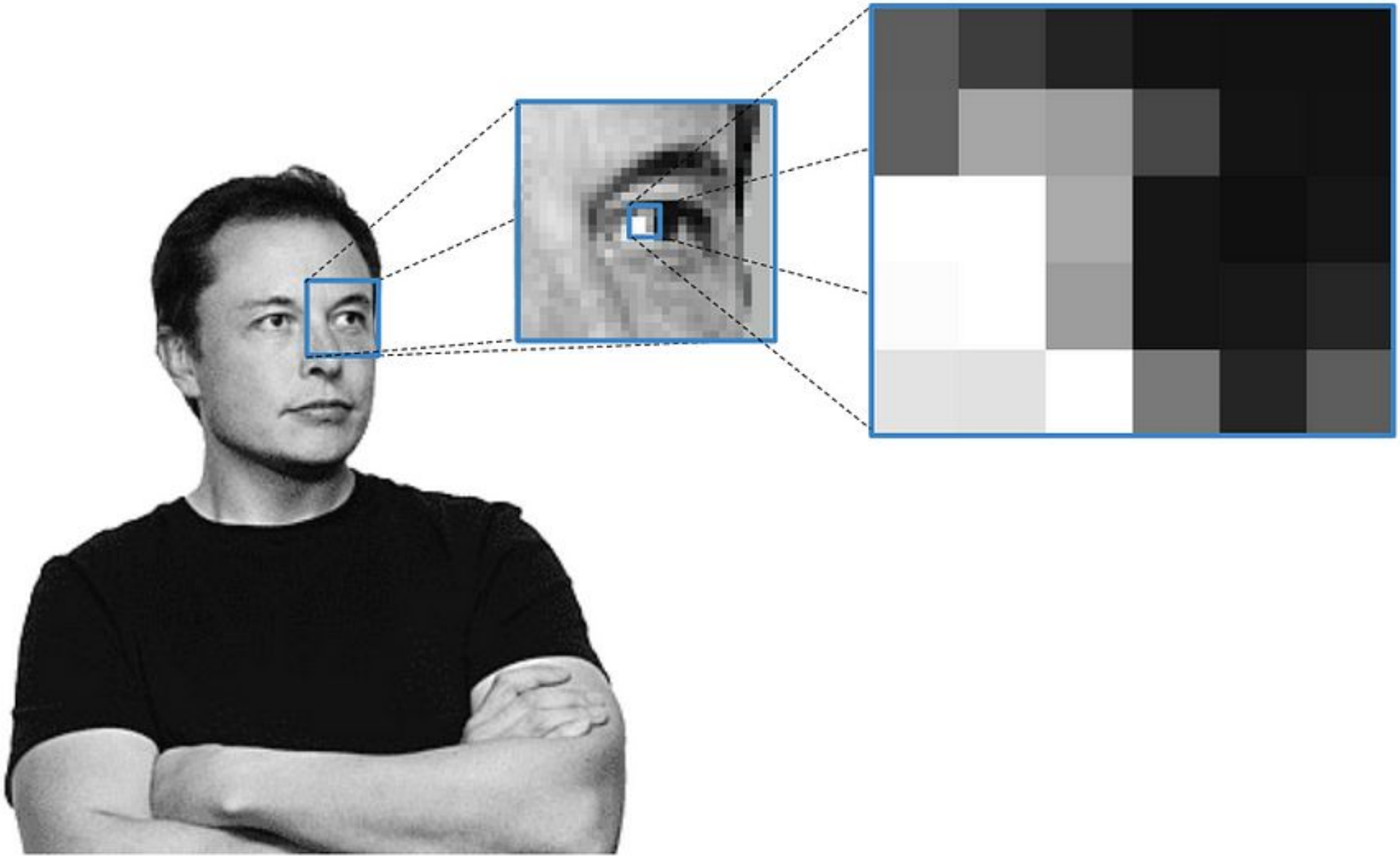
Image Analysis: The Fourier Transform allows for the analysis of image features in the frequency domain. By examining the amplitudes and phases of the frequency components, various image properties can be studied, such as texture analysis, edge detection, and shape recognition.

Image Enhancement: The Fourier Transform can be used for image enhancement by selectively amplifying or suppressing certain frequency components. This process is commonly known as frequency domain filtering or equalization. It can help enhance specific features or remove unwanted artifacts from an image.

CONVOLUTION AND FILTERING

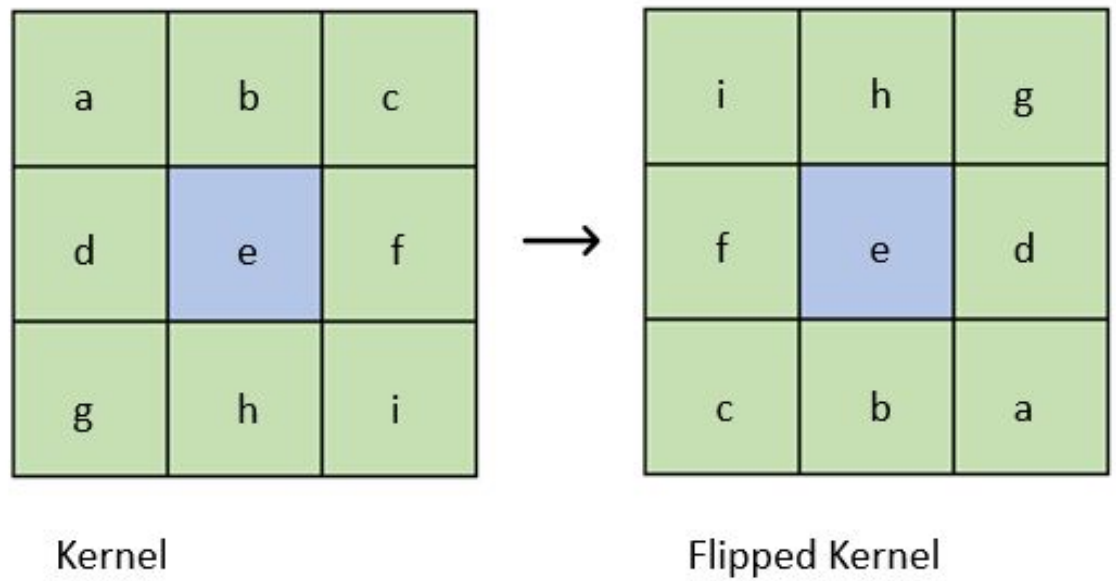
What is convolution?

- In basic terms, a convolution is a mathematical operation on two functions that produce a third function.
- Digital pictures are represented by pixel values. The typical format for storing pixels is by byte or eight bits. 8 bits can store 2^8 amount of information.
- Grayscale images have 1 channel that goes from a scale from 0 to 255 where 0 is black and 255 is white.
- Colored images typically have 3 channels and stored as 3 bytes: red, green, blue (RGB) values each ranging from 0 to 255 depending on intensity.
- Therefore, a picture can be represented as a matrix of values.

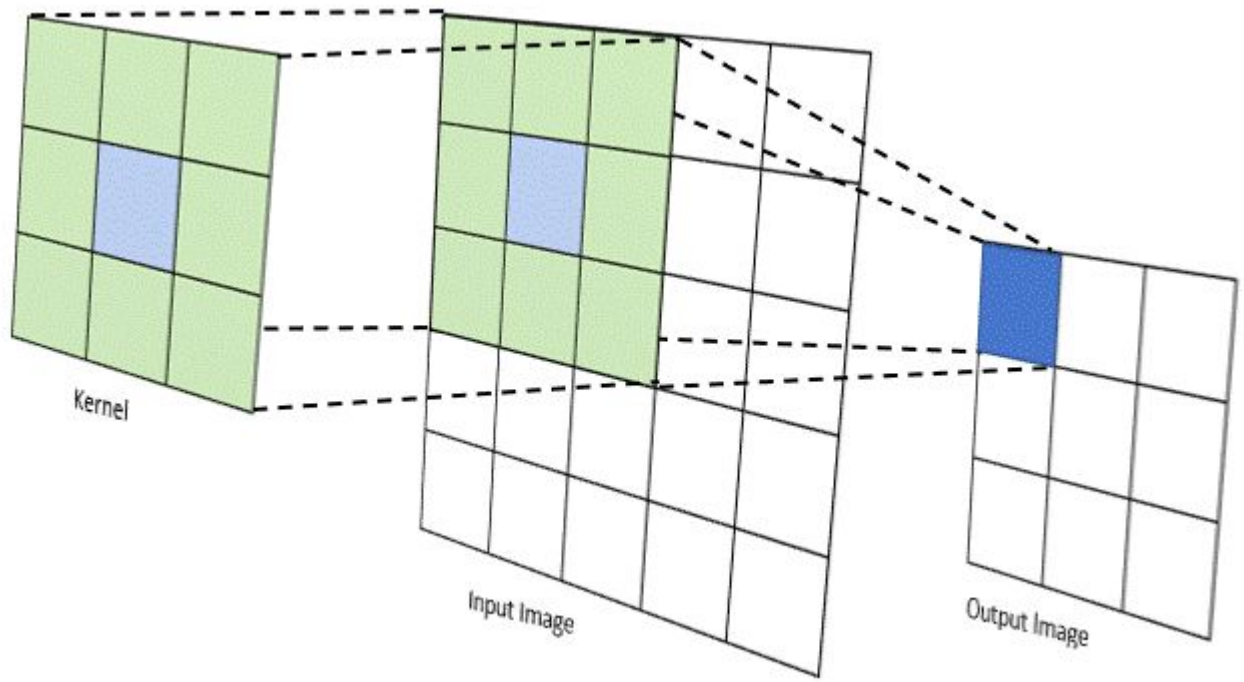


- In image convolution, involves a kernel, or matrix that is applied over the input image's pixels to generate an output image.
- The kernel size and values determine the effect the kernel has on the image. The dimensions of the kernel should be smaller or equal to that of the input image's.
- The kernel typically is square shaped and has an odd kernel size out of convenience.

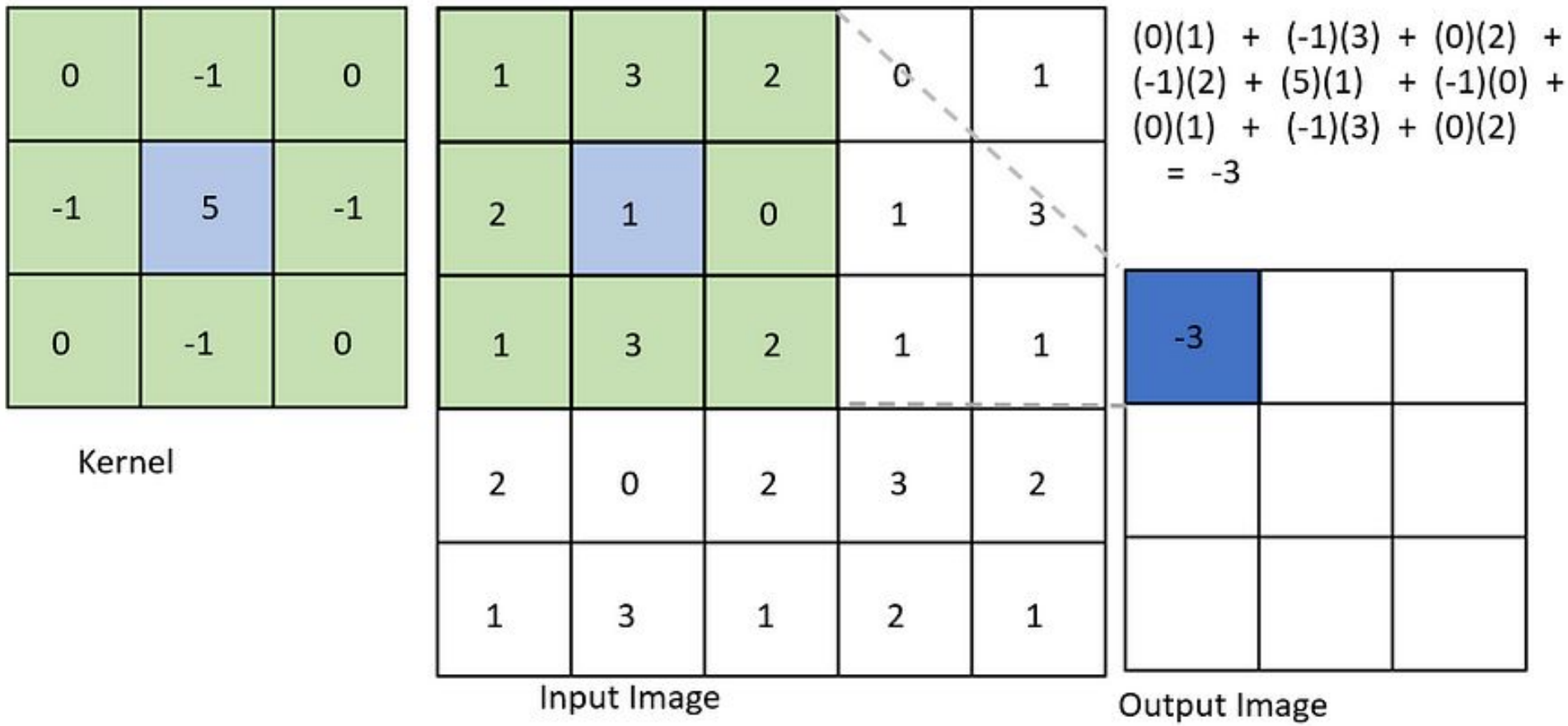
- How is the kernel applied in a convolution?
- The first step is that the kernel is flipped both horizontally and vertically. This is done by definition of a convolution.
- Using a non-flipped kernel would be doing a cross-correlation rather than a convolution.
- In the case of a symmetric kernel, the cross-correlation is equivalent to its convolution. Flipping or not flipping the kernel generally does not have a large impact on the resulting image visually.


a
b
c
d
e
f
g
h
i
i
h
g
f
e
d
c
b
a
Kernel
Flipped Kernel

- The animation below visually demonstrates how a 3x3 kernel is applied over a 5x5 input image generating a 3x3 output image. Note that the kernel slides along the input image.


Kernel
Input Image
Output Image

- The output image pixels are calculated by performing an element by element multiplication with the kernel and the covered section of the input image and then summing them up.
- Given an example kernel and input image, an example of the calculation is shown below with the first pixel.
- In the example, I intentionally used small numbers for ease of calculation.
- Additionally, its important to note that the output pixels of a convolution can yield values outside of 0–255.
- In some cases, it may be useful to normalize the results through Histogram Equalization or round the pixels to the nearest highest/lowest value.

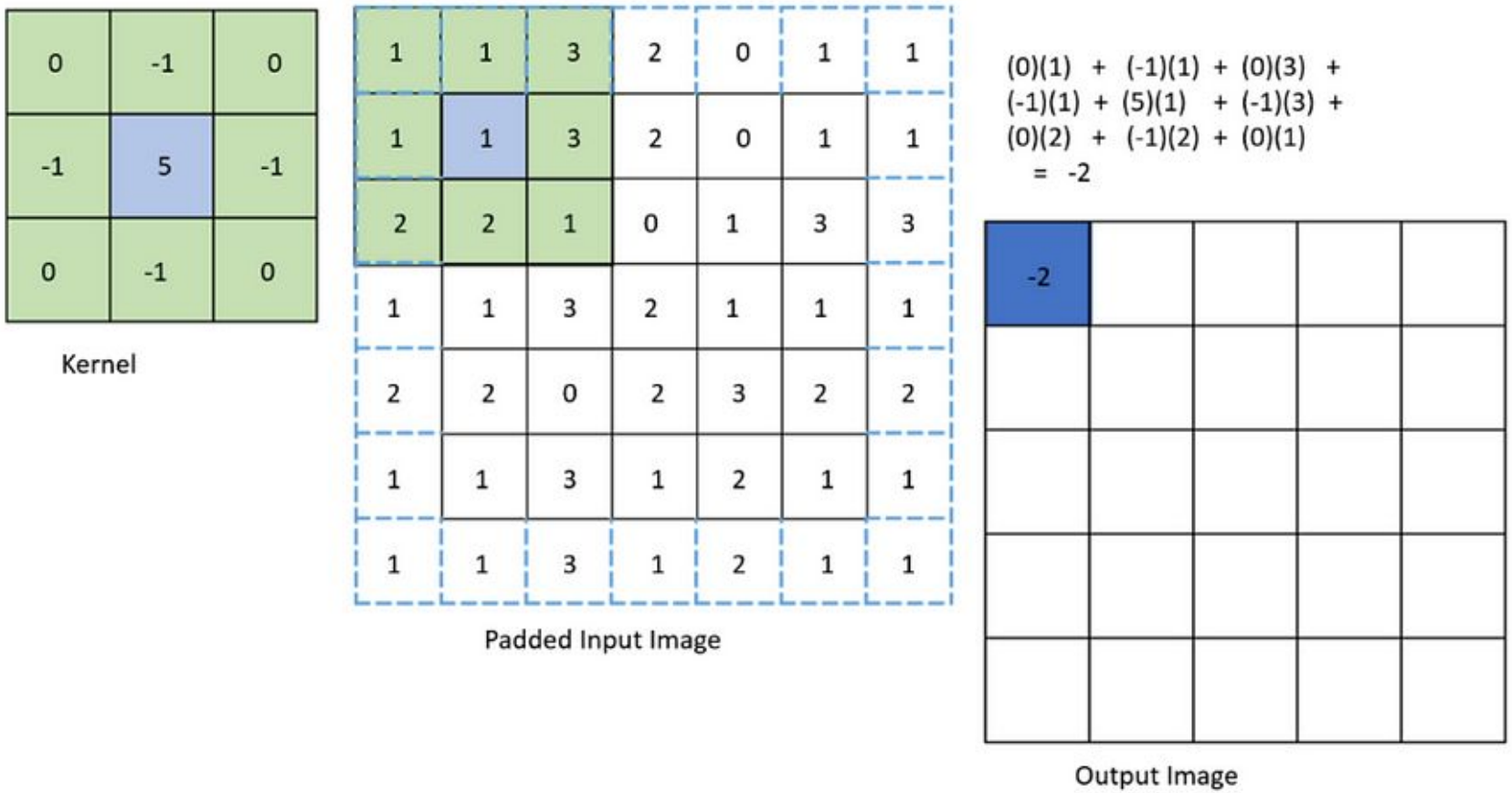

0 -1 0
-1 5 -1
0 -1 0
Kernel
1 3 2 0 1
2 1 0 1 3
1 3 2 1 1
2 0 2 3 2
1 3 1 2 1
Input Image
(0)(1) + (-1)(3) + (0)(2) +
(-1)(2) + (5)(1) + (-1)(0) +
(0)(1) + (-1)(3) + (0)(2)
= -3
-3
Output Image

- Mathematically, convolution in 2 dimensions is defined as follows:

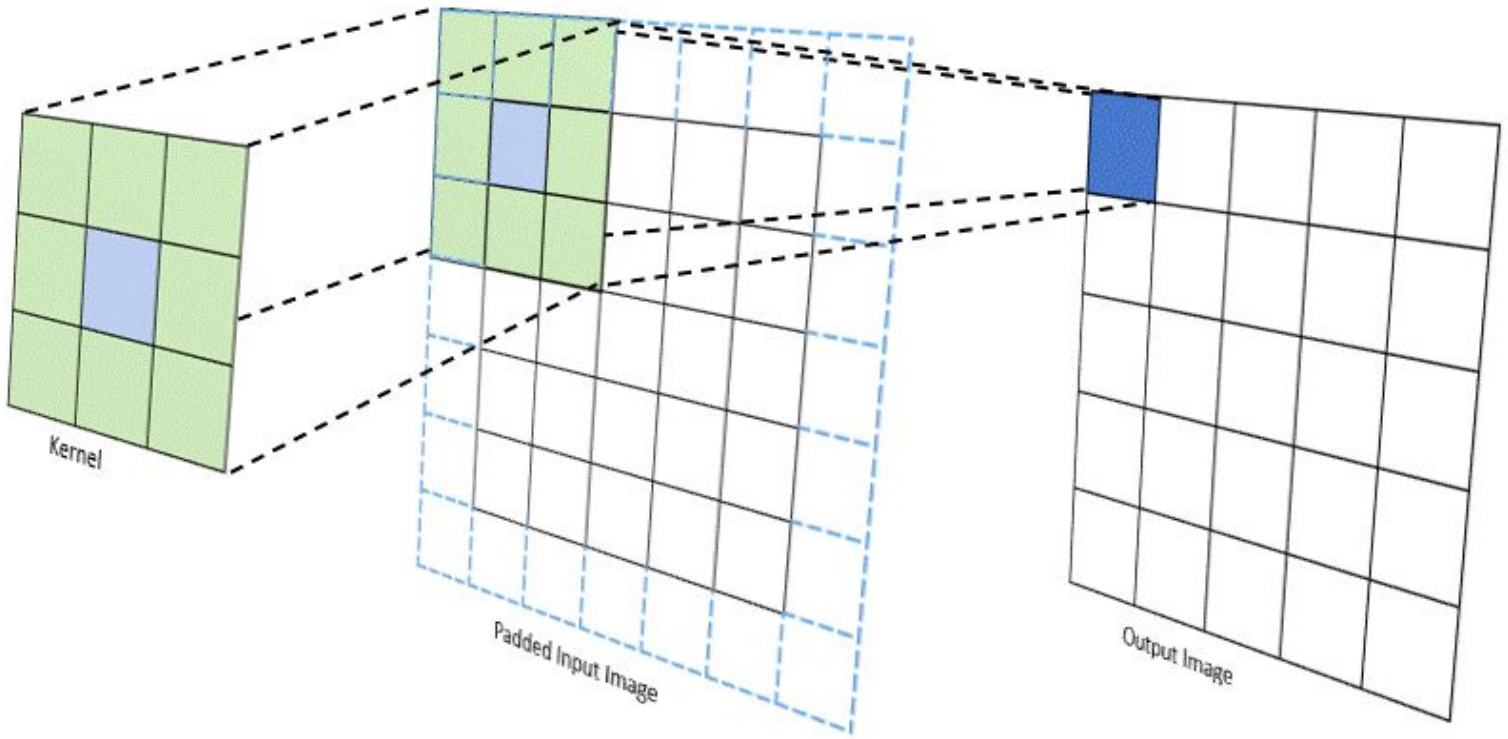
$$y(n,m) = \sum_k \sum_l x(k,l)k(n-k,m-l))$$

k, l represents the row, length indices of the kernel respectively. *x(n,m)* is the input and *y(n,m)* is the output images. *n,m* is the row, column indices of the input and output images.

- Notice that the output image size is smaller than the input image size.
- A larger kernel size would further decrease the output image dimensions.
- One way to fix this downsizing is to pad the input image. You can populate the padded image by extending the pixel values at the edge.
- Extending the edge pixels is one of many methods of padding.
- Below shows the input image padded by 1 pixel. The padded pixels are outlined in blue dotted lines.
-


0 -1 0
-1 5 -1
0 -1 0
Kernel
1 1 3 2 0 1 1
1 1 3 2 0 1 1
2 2 1 0 1 3 3
1 1 3 2 1 1 1
2 2 0 2 3 2 2
1 1 3 1 2 1 1
1 1 3 1 2 1 1
Padded Input Image
(0)(1) + (-1)(1) + (0)(3) +
(-1)(1) + (5)(1) + (-1)(3) +
(0)(2) + (-1)(2) + (0)(1)
= -2
-2
Output Image

The updated illustration with padding is shown below. Now, the output image has the same dimension as the original input image.



- The values of the kernels have differing effects on the output image. Using an example image of the dog shown below, here are some resulting images produced by the following kernels.
-

The following kernel sharpens the image.

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

This is a blurring kernel. The output pixels are determined by a combination of the central pixel and neighboring pixels.

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

THANK YOU