Parabola Volume 58, Issue 2 (2022)

Optimizing computer vision networks with fast Fourier transforms and the Convolution Theorem

Emmet Houghton[1]

1 Abstract

The incredible potential of artificial intelligence for the development of computer vision-based emerging technologies has made the optimization and application of convolutional neural networks (CNNs) the focus of significant research in recent years. This paper discusses the mathematics behind convolution and Fourier transformations and presents experiments that support the integration of a two-dimensional discrete Fast Fourier Transformation (FFT) layer into a CNN to minimize the computational costs of solving computer vision problems. The advantage of representing images in the frequency domain is that, by leveraging the power of FFT, convolutions can be calculated with a lower time complexity than traditionally possible. The experiments outlined in the paper validate the theoretical efficiency gained by FFT for images of varying sizes. The reduced computation time enabled by the application of these mathematics allows for the fitting of CNNs to datasets larger in both scale and resolution without requiring an increase in processing power. In this way, CNNs can be optimized to execute extensive image processing tasks in fields ranging from health care and transportation to robotics and cybersecurity, redefining the boundaries for the combined power of human and computer vision for the future of artificial intelligence solutions.

2 Introduction

Many emerging technologies of the last decade have leveraged an increasingly robust understanding of artificial intelligence along with the powerful and accessible processors of modern computers to solve complex computer vision problems. An integral part of these developments has been the convolutional neural network (CNN), a deep learning framework whose design is grounded in the mechanisms of the biological visual cortex as well as the structure of the traditional artificial neural network [6]. The CNN is the machine learning class that is largely responsible for many renowned innovations in image processing such as facial recognition, object detection, and AI-assisted medical diagnostics [13]. Naturally, maximizing the efficiency of CNNs is a pursuit of

[1]Emmet Houghton is a final-year student at Pingry School, New Jersey, USA.

interest to the research community. The primary purpose of this paper will be to outline and experimentally prove the mathematical theory that allows for convolutions to be calculated in the frequency domain and to be optimized using two-dimensional discrete fast Fourier transformations.

3 Theoretical Background

3.1 Two-Dimensional Convolution

The foundational mathematical concept utilized by CNNs for image processing is convolution. Convolution is a powerful mathematical operation integral to image filtering and the design of CNNs. As machines are unable to visually perceive images with light as humans do, an image must be converted to a matrix of numerical values in order for the computer to analyze characteristics of the image. Typically, these values represent brightness, RGB, or HSV values. Convolving this matrix of values representing an image is useful for various image processing purposes including sharpening, blurring, and edge detection.

As convolution requires two operands, a filter or "kernel" matrix must be defined in order to be applied to the original image matrix. Figures 1 and 2 present two examples of 3×3 filter matrices, taken from [3]:

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

Figure 1. 3×3 identity kernel

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Figure 2. 3×3 edge detection kernel

In order to complete a convolution using a kernel, each pixel of the source image is expanded to a matrix of equal size to the kernel by padding it with its adjacent pixels. Each pixel of the resulting "convolved" image is defined by the summation of the element-wise product of the kernel matrix and the expanded pixel matrix at that position. The identity kernel will not alter the source image, as each pixel is multiplied by 1 and unaltered by the surrounding pixels that are multiplied by 0. The edge detection kernel essentially assigns a numerical value to each pixel location in the output image based on how different the source image pixel at that location is relative to the pixels around it.

The general expression for each pixel of the convolved image with respect to an original image of size (x, y) and a 3×3 kernel is

$$g(x,y) = \omega(x,y) * f(x,y) = \sum_{\Delta x=-1}^{1} \sum_{\Delta y=-1}^{1} \omega(2+\Delta x, 2+\Delta y) f(x+\Delta x, y+\Delta y)$$

where $f(x, y)$ is the source image, $g(x, y)$ is the convolved image, and $\omega(x, y)$ is the kernel matrix.

Figure 3. Image of Prague

Figure 4. Convolved image using 3×3 edge detection kernel

Convolutions are computationally intensive calculations. The high resolution of modern images means that image matrices often have dimensions of more than 4,000 x 3,000 pixels, with an overall pixel count of over 12,000,000. Convolution calculations themselves are quite simple, but when twelve million discrete matrix multiplications are required, processing speed becomes an inhibiting factor to image processing. As such, convolution time is a significant barrier to efficient CNN training, because thousands of images must be processed to train the network before it sufficiently learns to recognize elements of images. In order to realize the full potential of computer vision AI, the optimization of image processing and other complex neural network algorithms is of critical importance. Consider autonomous driving, for example, during which images of the environment surrounding the vehicle must be processed in milliseconds in order to trigger steering, braking and evasive manoeuvrers. Fortunately, there are methods to significantly streamline convolution calculations. The mathematical theory underlying these optimization techniques will be presented in the following subsections.

3.2 Fourier Transformation

The Fourier Transform (FT) is an advanced calculus technique first employed by French mathematician Joseph Fourier in the early 1800s to translate signal information from the time domain into the frequency domain [14]. While studying the composition of advanced functions, Fourier developed the Fourier transform as a way to decompose these functions into an infinite sum of harmonic functions (such as sine or cosine). The concept can be most easily understood using an example of a sound wave made up of several different tones, or frequencies, such as a chord played on a piano. The output sound wave of the chord is the addition of the waves of each individual note played. In the diagram in Figure 5, the red sound wave depicts a chord that is the combination of the three purple component waves that represent each individual note within the chord [11]. While a listener might be able to recognize a chord as a whole, identifying

the individual notes that make up the composite sound wave (even with a graph of the wave in the time domain) is relatively difficult.

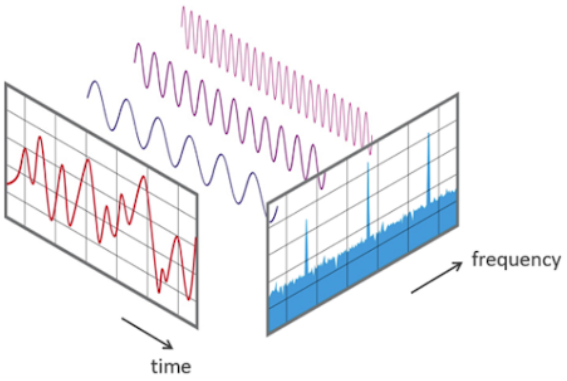


Figure 5. Visualisation of Fourier signal transformation [11]

The Fourier transform provides a way to perform such a calculation on the input sound wave and decompose it into the component waves of the individual notes making up the overall sound. The Fourier transform can be expressed as follows:

$$F(k) = \int f(x) e^{-i2\pi kx} dx$$

$f(x)$ is the input function and k is the output frequency.

In the example diagram above, the blue frequency function represents $F(k)$, the output of the Fourier transform. As k increases in the frequency domain, for each value k that matches a component note frequency in the time domain, the value of the Fourier transform, $F(k)$, equals the amplitude or strength of the individual note having frequency k.

Evaluating the Fourier transform to obtain the desired frequency function output requires some manipulation of the integral using Euler's formula, which states that

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

Euler's formula can be proven numerous different ways, for instance by using Taylor series for the exponential function e^x, the cosine function $\cos x$ and the sine function $\sin x$ [5]:

$$\begin{aligned} e^{ix} &= 1 + ix - \frac{x^2}{2!} - i\,\frac{x^3}{3!} + \frac{x^4}{4!} + i\,\frac{x^5}{5!} - \frac{x^6}{6!} - i\,\frac{x^7}{7!} + \cdots \\ &= \left(1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots\right) + i\left(x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots\right) \\ &= \cos x + i\,\sin x\,. \end{aligned}$$

Euler's formula can be substituted into the $F(k)$ formula by replacing $e^{-i2\pi kx}$ with $cos(-2\pi kx) + isin(-2\pi kx)$. The output of the new Fourier transform has a real component, $F_R(k)$, and an imaginary component, $F_I(k)$, and the magnitude of the output frequency wave is equal to $\sqrt{F_R(k)^2 + F_I(k)^2}$.

For image processing purposes, Fourier transforms must be implemented in their discrete form rather than the continuous form presented above. Further, since computers represent images in two-dimensional matrices, transforming an image into the frequency domain requires the transform calculation to be applied in two dimensions: once over each of the rows and then again for each of the columns. When applied to images, the Fourier transform separates the image into the component frequencies that can be used to construct the original pixel values in the space dimension [2]. Fourier-transformed images do not resemble their source images but maintain all image information. In fact, the original image can be restored from the Fourier transformed image in the frequency domain with the Inverse Fourier Transform (IFT):

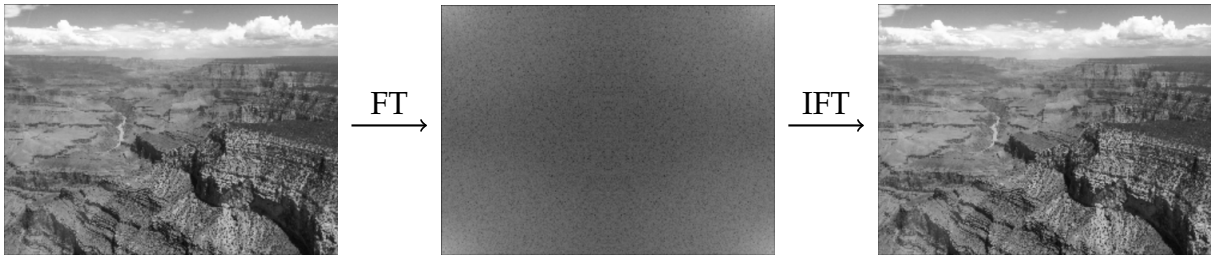


Figure 6. Image transformation into and out of the frequency domain

The formula for calculating a discrete 2D Fourier transform on an image of size (M, N) is as follows, where each pixel of the output image is represented by $F(u, v)$:

$$F(u,v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) e^{-2\pi i(\frac{xu}{M} + \frac{yv}{N})} .$$

The two-dimensional Fourier transform can be decomposed into individual one-dimensional Fourier transforms by manipulating the formula as follows:

$$F(u,v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) e^{-2\pi i(\frac{xu}{M})} e^{-2\pi i(\frac{yv}{N})} .$$

Now, notice that $e^{-2\pi i(\frac{xu}{M})}$ is a constant with respect to y and can be removed from the inner summation:

$$F(u,v) = \sum_{x=0}^{M-1} e^{-2\pi i(\frac{xu}{M})} \sum_{y=0}^{N-1} f(x,y) e^{-2\pi i(\frac{yv}{N})} .$$

The resulting expression demonstrates that the two-dimensional transform is equivalent to executing the one-dimensional transform for each column of the input matrix, and then again for each row of the input matrix [1].

3.3 Convolution Theorem

Critical to the optimization of convolutional neural networks presented in the next section is the Convolution Theorem, which combines the concepts of convolution and Fourier transformation. The Convolution Theorem states that the convolution of two matrices can alternatively be calculated by element-wise multiplication in the frequency domain. In other words, matrices can be convolved by taking the Fourier transform of each matrix, multiplying them together element by element, and then applying the inverse Fourier transform to the result:

The Convolution Theorem

$$\omega * f = F^{-1}[F(\omega)F(f)]\,.$$

Proof. Let x and y be vectors with convolution vector $z = x * y$. The discrete Fourier transform Z_k of z is

$$Z_k = \sum_n z_n e^{-2\pi ikn} = \sum_n (x * y)_n e^{-2\pi ikn} = \sum_n \Big(\sum_m x_m y_{n-m} \Big) e^{-2\pi ikn}\,.$$

Further, reverse the order of summation and extract the constant from the inside sum:

$$Z_k = \sum_m x_m \Big(\sum_n y_{n-m} e^{-2\pi ikn} \Big)\,.$$

Now, multiply by $e^{-2\pi ikm} = 1$ to align the index for the inside summation:

$$Z_k = \sum_m x_m e^{-2\pi ikm} \Big(\sum_n y_{n-m} e^{-2\pi ik(n-m)} \Big)\,.$$

The inside sum is the Fourier transform of y, which we will call Y. Note that Y is a constant with respect to m, and the outside sum is the Fourier transform of x, which we will call X. Thus,

$$Z_k = Y_k X_k\,.$$

This equation tells us that each individual element of the Fourier-transformed convolution vector is equivalent to the product of the corresponding elements of the original vectors in the frequency domain.

Finally, taking the inverse Fourier transform of both sides proves that

$$x * y = z = F^{-1}(YX) = F^{-1}\big(F(y)F(x)\big)\,. \qquad \square$$

The convolution of x and y is equal to the inverse Fourier transform of the vector produced by multiplying each corresponding element of their Fourier transforms [10]. Therefore, the convolution of two vectors can be calculated through three Fourier transforms (or, rather, two Fourier transforms and one inverse transform) instead of calculating the convolution directly. This fact is pivotal for the optimization of convolution, because, as will be discussed in the following section, there are significant inefficiencies to be leveraged in Fourier transform calculations. At this point, no efficiencies have been achieved due to that fact that the number of calculations required for both a convolution and a Fourier transform are on the order of N^2 for an N-dimensional square matrix.

4 Optimization Methodology

4.1 Fast Fourier Transforms

Fast Fourier Transforms (FFTs) were designed to exploit redundancies in Fourier transform calculations and allow for Fourier transforms to be calculated in a reduced number of operations (and, as a result, in much less processing time by computers). For ease of understanding, the following steps will outline the derivation of the discrete FFT in one dimension [9].

The 1D discrete Fourier transform $\mathbf{X}$ of any vector $\mathbf{x} = (x_0, \ldots, x_{N-1})$ has coordinates

$$X_k = \sum_{n=0}^{N-1} x_n e^{\frac{-i2\pi}{N}kn} .$$

Populating the resultant vector $\mathbf{X} = \{X_k\}$ requires evaluation for N values of k, and in turn a total of N^2 multiplications. Now, by letting $M = N/2$ and substituting for N, the above formula can be expressed as

$$X_k = \sum_{n=0}^{2M-1} x_n e^{\frac{-i2\pi}{2M}kn} .$$

Next, splitting the summation into its even and odd numbered terms produces

$$X_k = \sum_{n=0}^{M-1} x_{2n} e^{\frac{-i2\pi}{2M}k2n} + \sum_{n=0}^{M-1} x_{2n+1} e^{\frac{-i2\pi}{2M}k(2n+1)} .$$

Factoring the second term and simplifying the exponents leaves us with

$$X_k = \sum_{n=0}^{M-1} x_{2n} e^{\frac{-i2\pi}{M}kn} + \sum_{n=0}^{M-1} x_{2n+1} e^{\frac{-i2\pi}{M}kn} e^{\frac{-i2\pi}{2M}k} .$$

Now, realizing that the main terms are simply the Fourier transforms of the even and odd elements of the original vector, X_k, can be further simplified as

$$X_k = X_k^{even} + X_k^{odd} e^{\frac{-i2\pi}{2M}k} . \tag{1}$$

And finally, using the facts that

$$e^{\frac{-i2\pi}{M}(k+M)n} = e^{\frac{-i2\pi}{M}kn} e^{-i2\pi n} = e^{\frac{-i2\pi}{M}kn} 1^n = e^{\frac{-i2\pi}{M}kn} ,$$

we see that

$$X_{k+M}^{even} = X_k^{even} \qquad \text{and} \qquad X_{k+M}^{odd} = X_k^{odd} ,$$

so

$$e^{\frac{-i2\pi}{2M}(k+M)} = e^{\frac{-i2\pi}{2M}k} e^{-i\pi} = e^{\frac{-i2\pi}{2M}k}(-1) = -e^{\frac{-i2\pi}{2M}k} .$$

We can write the following by substituting $k + M$ in for k in (1) and find

$$X_{k+M} = X_k^{even} - X_k^{odd} e^{\frac{-i\pi}{M}k}. \tag{2}$$

Transforming the original one-dimensional vector can thus be broken into two tasks. Each element in the first half of the vector can be calculated with two Fourier transformations of length $N/2$ by equation (1). For the second half of the vector, however, the transforms computed for the first half can be reused according to equation (2). Even better, the process is recursive, because the first half of the vector can be split into two segments for further optimization. Thus, while a standard Fourier transform for a vector of N elements requires a number of computations proportional to N^2, the FFT reduces the requirement to the order of $Nlog_2N$, a powerful reduction in processing time that actually grows in efficiency as the image size, N increases.

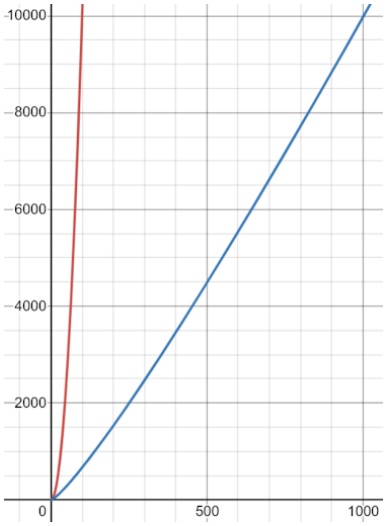


Figure 7. Number of computations required to execute algorithm of order N^2 (red) versus $N \log_2 N$ (blue)

Naturally, the Fast Fourier Transform can be applied in the context of convolution by the Convolution Theorem. As such, modern computers can calculate convolutions using 3 FFTs and one element-wise matrix multiplication, which for large N can reduce processing speeds by greater than 90% [10].

4.2 Experimental Validation

The efficiency of FFT-based convolution can be observed experimentally. To demonstrate the power of these mathematics, images of various resolutions were convolved with a 5×5 edge detection kernel (Figure 8) both with traditional, two-dimensional convolution and using SciPy's FFT/IFFT algorithms in python [12]. A code fragment is also attached below in Figure 9.

$$\begin{bmatrix} 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & -2 & -1 & 0 \\ -1 & -2 & 16 & -2 & -1 \\ 0 & -1 & -2 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{bmatrix}$$

Figure 8. A 5×5 edge detection kernel

```
# Kernel
kernel = np.array([[-1,-1,-1], [-1, 8,-1], [-1,-1,-1]])

# Traditional convolution
start = time.time()
output = twodconvolve(image, kernel, padding=2)
print("2D Convolution Time:", time.time() - start, "ms")
cv2.imwrite('Convoluted.jpg',output)

# FFT-based convolution
l1 = tf.shape(image)[0]
l2 = tf.shape(image)[1]
start = time.time()
im_fft = tf.signal.rfft2d(image, fft_length=[l1,l2])
kernel_fft = tf.signal.rfft2d(kernel, fft_length=[l1,l2])
im_convolved = np.array(tf.signal.irfft2d(im_fft * kernel_fft, [l1,l2]))
print("FFT Convolution Time:", time.time() - start, "ms")
cv2.imwrite('FFTConvoluted.jpg', im_convolved)
```

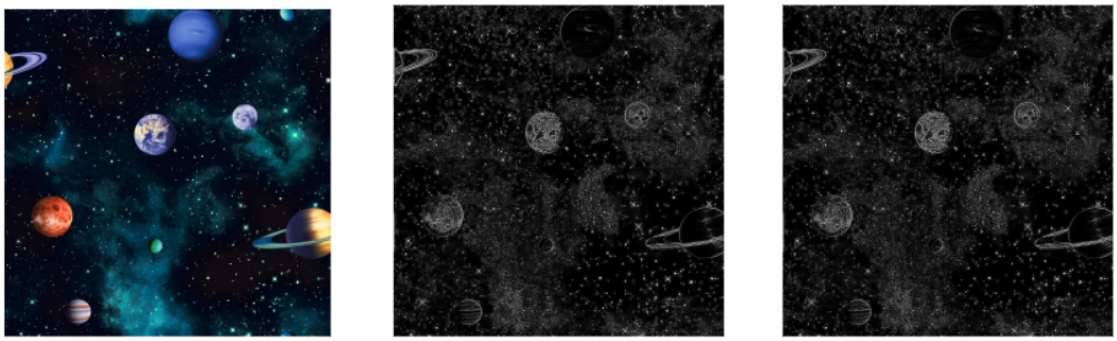
Figure 9. Python code to calculate and time traditional and FFT-based two-dimensional discrete convolution.

While the different techniques produced identical matrices, the frequency-domain convolution generated the final image faster by leveraging the efficiency of FFT in both cases. Also note that FFT yielded larger efficiency gains in convolving the larger image.

(a) (b) (c)

Figure 10.

(a) Original image of dimensions 500×500 pixels
(b) Traditionally convolved image computed in 0.942 seconds
(c) FFT-convolved image computed in 0.193 seconds



(a) (b) (c)

Figure 11.

(a) Original image of dimensions 1000×1000 pixels
(b) Traditionally convolved image computed in 5.424 seconds
(c) FFT-convolved image computed in 0.770 seconds

5 Conclusion

The optimization of machine learning discussed in this paper through the theoretical lens of mathematics illustrates the ability of mathematics to transcend time and fields of study. Even the Fast Fourier Transform, which was developed in 1965, predates the rise of computer science and machine learning [14]. Today, FFT is hailed as one of the greatest mathematical algorithms of the twentieth century for its extensive applications for tasks like audio, image, and video processing [7]. This paper has both theoretically and experimentally proven that image convolution can be substantially optimized with FFT according to the Convolution Theorem and thus supports the integration of an FFT layer into convolutional neural networks. The simulations conducted in Section 4 indicate that FFT-based convolution can accelerate image filtering by incredible margins, and that these efficiency gains increase for larger kernel and image sizes. These results are a testament to the power of FFT and to the tangible value of seemingly abstract mathematical discovery, and the ubiquitous applications of convolutional neural networks for solving computer vision problems in diverse fields make this research critical to the technological transformation of modern society.

References

[1] E. Agu, *Digital Image Processing (CS/ECE 545) Lecture 10: Discrete Fourier Transform (DFT)*, https://web.cs.wpi.edu/ emmanuel/courses/cs545/S14/slides/lecture10.pdf, last accessed on 2022-03-28.

[2] R. Fisher, S. Perkins, A. Walker and E. Wolfart, *Hypermedia Image Processing Reference (HIPR)*, John Wiley & Sons Ltd., Chichester, 1997.

[3] K. Gerding, Image kernels, *Medium*, 8 June 2016, https://medium.com/@kgerding/image-kernels-2f8a36087b75, last accessed on 2022-03-28.

[4] A.C. Gilbert, P. Indyk, M. Iwen and L. Schmidt, Recent developments in the sparse Fourier transform: A compressed Fourier transform for big data, *IEEE Signal Processing Magazine* **31** (2014), 91–100.

[5] O. Hansha, Understanding Euler's Formula, *Medium*, 28 November 2017, https://ozanerhansha.medium.com/understanding-eulers-formula-888e5f58f559, last accessed on 2022-03-28.

[6] D.H. Hubel and T.N. Wiesel, Receptive fields, binocular interaction and functional architecture in the cat's visual cortex, *The Journal of Physiology* **160(1)** (1962), 106–154.

[7] H. Knight, Leaner Fourier transforms, *MIT News*, 11 December 2013, https://news.mit.edu/2013/leaner-fourier-transforms-1211, last accessed on 2022-03-28.

[8] Lancashire Wallpaper & Paint Co., *Cosmos-solar-system-668100*, 2017, https://lancashirewallpaper.co.uk/products/cosmos-space-wallpaper-668100, last accessed on 2022-03-28.

[9] D. Marshall, *The Fast Fourier Transform Algorithm*, https://users.cs.cf.ac.uk/Dave.Marshall/Vision lecture/node20.html, last accessed on 2022-03-28.

[10] V. Nair, M. Chatterjee, N. Tavakoli, A.S. Namin and C. Snoeyink, Fast Fourier transformation for optimizing convolutional neural networks in object recognition, https://arxiv.org/abs/2010.04257v1, last accessed on 2022-03-28.

[11] NTi Audio, *Let's Clear Up Some Things About FFT... – Part 1 of 2: The Basics*, https://www.nti-audio.com/en/news/lets-clear-up-some-things-about-fft-part-1, last accessed on 2022-03-28.

[12] SciPy, *Legacy discrete Fourier transforms*, https://docs.scipy.org/doc/scipy/reference/fftpack.html, last accessed on 2022-03-28.

[13] X. Tang, J. Peng, B. Zhong, J. Li and Z. Yan, Introducing frequency representation into convolution neural networks for medical image segmentation via twin-kernel Fourier convolution, *Computer Methods and Programs in Biomedicine* **205** (2021), 106110.

[14] R. Thummalapalli, Fourier transform: Nature's way of analyzing data, *Yale Scientific*, 1 December 2020, https://www.yalescientific.org/2010/12/fourier-transform-natures-way-of-analyzing-data/, last accessed on 2022-03-28.