

L'Aquila 2023

Abstract

Forecasting financial time series has been classified as one of the most challenging problems in the last decade due to its non-stationarity and non-linear properties. On one hand, statistical techniques have been found incapable of accurately predicting financial time series. On the other hand, machine learning techniques have achieved remarkable results, but they do not provide an explicit way of handling the non-stationarity property of financial time series. The proposed approach leverages the capabilities of signal processing decomposition techniques to address the non-stationarity property of financial time series. The signal decomposition technique employed in this work is iterative filtering (IF), which generates intrinsic mode functions (IMFs). These generated IMFs, along with the original signal, are used to produce a time-frequency representation of the financial time series, called IMFogram. Two types of data, namely the IMFs and IMFogram, are utilized to train a fusion neural network for predicting the financial time series. One entry component of the fusion neural network is an artificial neural network (ANN) taking the IMFs as input. The other entry component of the fusion neural network is a convolutional neural network (CNN), which takes the IMFogram as input. The outputs of the ANN and the CNN are concatenated for a regression task. We show the application of this newly developed approach to financial data, NASDAQ series to be precise. And we report its performance in different scenarios of boundary conditions.

Keywords

Artificial neural network (ANN), Convolutional neural network (CNN), Fusion Neural Network, Iterative Filtering (IF), Intrinsic Mode Functions (IMFS), IMFogram, symmetric extension, asymmetric extension, Time series,...

I declare that I wrote the diploma thesis *Financial time series based on innovative Machine Learning Signal Processing approach* independently under the guidance of *Antonio Cicone* using the literature included in the list of references.

Reagan Tshiangomba Kasonsa

I would like to express my gratitude for having the opportunity to work under the supervision of Professor Antonio Cicone. His guidance has been of great importance, always ready to discuss and clarify any misunderstood subjects. Once again, I thank him for his supervision.

Reagan Kasonsa Tshiangomba

Contents

1	Introduction	11
1.1	Literature review	11
1.1.1	Literature review with respect to statistical techniques	11
1.1.2	Literature review concerning Machine learning	12
1.1.3	Literature review regarding hybrid model	13
1.1.4	Literature review about non-stationary signal processing	14
1.2	Proposed work	15
2	Iterative Filtering and IMFogram	16
2.1	Iterative Filtering	16
2.2	IMFogram	26
3	Artificial neural network and Convolutional Neural Networks	27
3.1	Neural Network	27
3.2	Convolutional Neural Network	30
3.2.1	Convolutional Layer	31
3.2.2	Non Linear Activation Layer	31
3.2.3	Batch Normalization Layer	32
3.2.4	Pooling Layer	32
3.2.5	Dropout Layer	32
3.2.6	FC Layer	32
4	Proposed Approach	33
4.1	Artificial neural network (ANN)	33
4.2	Convolutional neural network (CNN)	35
4.3	Fusion neural network (FNN)	38
4.4	Back propagation of FNN	40
5	Datasets and the simulation results	44
5.1	Datasets	44
5.2	Simulation Results	50
5.2.1	Experimental setting	50
5.2.2	Evaluation criteria	51
5.2.3	Prediction of FTS based on asymmetric extension	51
5.2.4	Prediction of FTS based on symmetric extension	52
5.2.5	Prediction of FTS based on asymmetric and symmetric extension	52
5.2.6	Prediction of FTS based on symmetric and asymmetric extension	54
6	Conclusion and future work	55

1 Introduction

Financial Time Series (FTS) describes the evolution of the stock market over time. FTS is affected by microeconomic and macroeconomic factors, which makes it difficult to predict. These factors make FTS non-linear, noisy, non-stationary, and time-dependent. It is really difficult to understand the mechanism underlying FTS. The processing of FTS requires new techniques for handling non-stationarity and noise since the existing techniques, such as Fast Fourier Transform (FFT), autoregressive integrated moving average (ARIMA), generalized autoregressive conditional heteroskedasticity (GARCH) volatility [1], smooth transition autoregressive model (STAR) [2], and hidden Markov model (HMM) [3], are not well-suited for FTS. The limitations of traditional time series forecasting models arise from their reliance on the assumptions of stationarity and linearity, which often do not hold true in reality. Advances in financial transactions and information systems have led to a substantial increase in available data, enabling practitioners to make more accurate Financial Time Series (FTS) predictions [4, 5, 6]. The primary motivation behind forecasting FTS lies in the significant impact that even a slight improvement in accuracy can have on the profits of financial institutions and individuals engaged in financial transactions. Hence, the pursuit of the ability to predict FTS with substantial accuracy is considered highly valuable.

1.1 Literature review

In this section, we explore the literature with a specific focus on feature extraction and prediction accuracy of FTS. The difficulty of extracting features and the low prediction accuracy in FTS forecasting are major problems. Several techniques and critical work have been proposed to improve the forecasting of FTS. The practical techniques for forecasting FTS can be grouped into three categories: models based on statistical techniques, those based on Machine Learning (ML), and those focusing on hybrid techniques. The state-of-the-art ML bases approached achieved significant prediction accuracy in recent years and performs better than statistical techniques, especially in market predictions, risk management, and derivative pricing [7].

1.1.1 Literature review with respect to statistical techniques

The process of forecasting a time-series is intimately related to the specification of a model. This model is in fact a statistical formulation of the dynamic relationship between the observed information and the variable related to these observations. Linearity among normally distributed variables is the cornerstone of traditional statistical models [8]. One of the popular and widely used time series models is the Auto-Regressive Integrated Moving Average (ARIMA) [9, 10]. As a statistical model ARIMA can implement various exponential smoothing models, has the advantages of accurate forecasting over short period of time and easy to implement, but suffer to correctly predict a time series generated by a non-linear source. It is well documented that real world systems are often generated by nonlinear sources[11, 10].

The fact that non-linearity is an intrinsic property of real world times series lead to the formulation of several class of nonlinear models in the literature to overcome the linear limitation of the time series models. Among these models there are te bilinear model [12], the threshold autoregressive (TAR) model [13], the autoregressive conditional heteroscedastic (ARCH) model [14], general autoregressive conditional heteroscedastic (GARCH) [15], and chaotic dynamics [16].

1.1.2 Literature review concerning Machine learning

The popularity of FTS forecasting among ML researchers has been growing in the last 40 years. Researchers have built several ML models, and a tremendous number of studies have been published. The comparison of ML techniques with respect to different financial applications, including stock market prediction, has been studied in [17]. The use of evolutionary computation (EC) and artificial neural networks (ANN) has gained a lot of consideration in a number of papers. Chen proposed a wonderful book on genetic algorithms (GAs) and genetic programming (GP) in computational finance based on EC [18]. In [19] [20] the authors surveyed the use of Multi-objective Evolutionary Algorithms (MOEAs) to tackle financial applications comprising FTS forecasting. In [21], the potential use of ANN was appraised and implemented for the forecast of stock price and other financial applications. In [22], it has been highlighted that ANN is established as a well-known method in financial applications, including FTS forecasting, and the improvement of their functioning and the amelioration of our understanding of this marvelous area require more additional research.

The use of text mining for financial applications and FTS prediction was the concern of some authors. Using text mining for the stock or forex market, Nassirtoussi et al. surveyed how sentiment in social media and online news could determine the predictability of financial markets and cause huge gains or losses [23]. In [24], the author used textual sentiment for time series forecasting and trading strategies. A state-of-the-art survey of FTS forecasting and FOREX rate prediction was provided in [25].

FTS has gained a lot of attention compared to other financial applications, and the use of ML provides flexible frameworks to tackle FTS forecasting. A huge amount of surveys has been published for the forecast of FTS studies based on several soft computing techniques at different times. The same techniques were used to summarize and visualize stock market data for individuals and financial institutions to gain useful information about market behavior and to make investment decisions [26, 27, 28, 29, 30, 31]. In the past ten years, ML has developed different novel techniques to analyze useful features from a large amount of data [32]. The main aim of those techniques is to model complex real-world data by extracting robust features that capture the pertinent information [33].

In order to extract robust features from data more effectively, the use of deep learning (DL) algorithms is necessary. DL is a subset of machine learning (ML) that processes complex data using multi-layered neural networks, mimicking the biological structure of the human brain. It is divided into three stages. In the first stage, called the input layer, neurons receive the input data, process it, and transfer the result to the next stage, known as the hidden layer. The hidden layer then processes the received result and sends it to the third stage, called the output layer [34]. In the literature, DL is predominantly used in applications such as image processing [35, 36, 37], natural language processing [38, 39], healthcare [40], and more.

DL has made considerable advancements over the past decade, and important details are surveyed in [41]. The introduction of DL in the financial community has proven to be significant, especially for FTS prediction, and has resulted in numerous high-quality publications [42]. Various types of DL models have been proposed in the literature, including Deep Multilayer Perceptron (DMLP), Recurrent Neural Network (RNN), Long-Short Term Memory (LSTM), Convolutional Neural Network (CNN), Restricted Boltzmann Machines (RBMs), Deep Belief Network (DBN), Autoencoder (AE), and Deep Reinforcement Learning (DRL) [41, 43].

The first work out ANN is DMLP; its architecture consists of principally three layers: the input, hidden, and output. The hyperparameters of DMLP are the number of neurons appearing in each layer and the number of layers in the network [44, 41, 42]. DMLP can solve regression and classification problems easily just by modeling the input data [45].

DL has several types of architectures, and another commonly used one is RNN. RNN is often used for time series or language and speech recognition. The analysis of time series data is often done using RNN in various fields such as handwriting recognition, speech recognition, etc. [41, 44].

RNN is best fitted for learning long-term dependencies. One issue highlighted in the literature is that when knowledge is stored for long time periods, it is really complicated to learn with RNN [46]. LSTM is another ANN architecture that solves this issue. LSTM is a version of RNN with the property of remembering both short and long-term values. LSTM appears to be the most used DL architecture, especially for tackling time-series data and FTS analysis [47]. LSTM introduces cells to store data indefinitely, and in this way, the architecture can decide to remember or forget. LSTM is efficient in machine translation [48, 49].

Apart LSTM, Convolutional Neural Network (CNN) is another type of Deep Neural Network (DNN) architecture that uses convolutional layers based on convolution operations between the filter, or weight matrix, and the input, with the ability to act directly on the raw inputs. CNN, as a cutting-edge DNN architecture, is frequently used for computer vision or image processing-based feature extraction such as image classification.

Due to its complex architecture, CNN can learn filters that are capable of recognizing specific features in the input data. This ability has currently attracted considerable attention. In [50], a one-dimensional CNN is used to extract robust features from one-dimensional time series. In the work by Mittel and Roni [1], a novel time-series model using a convolutional neural network architecture was introduced. Specifically, this innovative model adopts a fully convolutional network (FCN) structure employing causal filtering operations, enabling the output signal rate to match that of the input signal. Additionally, drawing inspiration from the undecimated wavelet transform, the authors put forth an undecimated variant of the FCN, referred to as the undecimated fully convolutional neural network (UFCNN).

1.1.3 Literature review regarding hybrid model

We can remark from the literature that the chronological order of data is the essential consideration of models based on 1-dimensional Convolution operation, ignoring other factors. Another major problem arising in 2-dimensional CNN models is the lack of handling the non-stationarity side of FTS accordingly. To solve this issue, some authors prefer to use hybrid systems or focus on not completely abandoning statistical methods. A hybrid combination of statistical models and ML techniques is used to predict FTS in [51, 52]. A work by Zang, which was the combination of ARIMA and Support Vector Machine (SVM), appeared to be a superior technique for improving forecasting accuracy when applied to real datasets [53]. The combination of ANN and genetic algorithm (GA) using evolutionary search to determine the spatial features of the time series was used in [54]. In [55], the authors developed a tool to address in-depth analysis of the stock market. The developed tool combines the Hidden Markov Model (HMM), ANN, and GA. In [56], the authors proposed a fusion of Empirical Mode Decomposition (EMD) and ANN for the purpose of forecasting and applied these techniques to the Baltic Dry Index. In [57], the authors developed a hybrid end-to-end approach combining EMD and Factorization Machine-based Neural Network

(EMD2FNN) to predict stock market trends.

Afterwards, they proposed another hybrid version, a two-stage approach, namely IF2FNN, combining Iterative Filtering (IF) and Factorization Machine-based Neural Network, to predict (including short-term and long-term predictions) and recover the general types of time series [58]. Then, they came up with the idea of combining IF methods with CNN for automatic feature learning for FTS forecasting, called IF2CNN [59].

Plunging into the literature studies of FTS forecasting, we can understand that hybrid prediction approaches can improve prediction performance by overcoming the shortcomings of single models, handling model uncertainty, and increasing generalization ability simultaneously [60]. In [61], the study on the effectiveness of several ML techniques and their one-step prediction methods of a series of financial data discovered that the hybrid model obtained combining some linear statistical models and nonlinear ML algorithms is powerful in predicting the future value of sequence data, especially in the future direction of the sequence. Notwithstanding, hybrid models, despite performing well, have some limitations with respect to time complexity and computational efficiency [62].

1.1.4 Literature review about non-stationary signal processing

Real-world systems often exhibit nonlinear characteristics [11]. This phenomenon is evident in various real-world signals, including FTS, machine vibrations, speech, radar and sonar acoustic waves, seismic acoustic waves, and biomedical signals such as the electrocardiogram (ECG) or neonatal seizures. Additionally, non-stationarity is observed in the impulse response of wireless communications channels, biological signals, vocals in speech, notes in music, and engine noises [63, 64].

In the realm of biological signals, a diverse range of sensors has been developed to measure biosignals reflecting various underlying physiological phenomena. For instance, gyroscopes and accelerometers are utilized to measure pathological and physiological tremor signals [65], accelerometers are employed for monitoring cardiac mechanical vibrations [66], infrared sensors are utilized for monitoring respiratory motion [67], and standard electrodes are employed for measuring electrical activity in the brain and heart [68, 69]. All the mentioned physiological signals are non-stationary due to the complex nature of biological systems [70].

To address non-stationary signals, the scientific community has introduced various signal decomposition techniques. Huang proposed the empirical mode decomposition (EMD) for analyzing nonlinear and non-stationary data. EMD breaks down complex datasets into a finite number of intrinsic mode functions (IMFs) that undergo well-behaved Hilbert transforms [71]. However, EMD has shown instability under perturbations. To overcome this challenge, Huang et al. introduced the ensemble empirical mode decomposition, which involves adding noise to the original signal. EMD is then applied to the noisily obtained signal to generate IMFs. This process is repeated for different realizations of noise, and the final IMFs are obtained by averaging those from different noisy signal realizations [72]. Another signal decomposition method inspired by EMD is iterative filtering (IF). Instead of computing the average of the upper and lower envelopes of the signal, IF applies a filter to the signal and subtracts the filtered signal from the original [73].

In the quest for methods to analyze non-stationary signals, the synchrosqueezed transform was

proposed in [74], known for its robustness to bounded perturbations and Gaussian white noise. Another approach, presented in [75], is the variational mode decomposition (VMD), which is entirely non-recursive, enabling concurrent extraction of modes. However, VMD's reliance on the narrow-band property of signal modes limits its effectiveness in analyzing wide-band nonlinear chirp signals (NCSs). Addressing this limitation, [76] proposed an alternative method called variational nonlinear chirp mode decomposition (VNCMD) [77].

Singular spectrum analysis (SSA) is yet another signal decomposition technique designed to break down signals into interpretable and physically meaningful components. In [78], the sliding SSA method was introduced, providing both theoretical and practical insights into the separability of SSA [77].

1.2 Proposed work

This work is a contribution to the community of FTS prediction, since it provides the use of two different type of datasets as input of a fusion neural network to predict the FTS. The proposed framework can serve as baseline for the combination of other sophisticated neural network architectures for time series prediction.

This work is based on innovative machine learning signal processing approach. It takes advantage on the hybrid framework combining the Iterative Filtering (IF) [79, 80, 81], IMFogram [82, 83], ANN and the CNN. IF has a tremendous advantages of reducing the influence of noise and non-stationarity of the times series in some extent. Furthermore, signal processing approach provides two type of datasets, (1) the intrinsic mode functions (IMFs) through the IF algorithm and an (2) IMFogram time-frequency representation of the time series.

Given these two types of datasets, the goal of this work is to predict the financial time series. To achieve this goal we start producing an IMFs decomposition of a FTS via IF the sum of all the IMFs is the underlying FTS itself. Then, using the IMFogram algorithm [82, 83], we produce a time-frequency representation of the FTS.

Given the two datasets, we build a deep neural network that concatenates a ANN (taking as input all the IMFs) and a CNN (taking as input the IMFogram). The output of the ANN is concatenated with the output of the CNN which is flatten to have the same dimension as the output of the ANN. The squared loss for regression purpose and the back-propagation are applied to learn from the data.

The rest of the work is organized as follows: In Section 2, we present a general review of IF and IMFogram, whereas in Section 3 we provide a mathematical definition of neural networks and CNN. Section 4 details the main approach used to predict the underlying FTS. In Section 5, we present the two type of datasets and the simulation results. Section 6 presents a brief conclusion and key points of open issues for further studies.

2 Iterative Filtering and IMFogram

2.1 Iterative Filtering

Data and signal analysis are ubiquitous nowadays; therefore, creating tools to tackle them is of great importance. Data is rarely perfect and can be subject to various sources of noise. It can also be non-linear due to the complex, non-linear behavior of real-world phenomena, and natural processes. Additionally, data can be non-stationary, which means that the mean, variance, and other statistical properties do not remain constant over time. Non-stationarity and non-linearity are common and important features of many real-world datasets, and it is important for researchers and analysts to be aware of this when developing models and interpreting results. Furthermore, when the data is non-stationary and non-linear, the decomposition of the signal and the extraction of features are very challenging.

The Fourier spectral analysis and wavelet have been found not well suited tackling non-linear and non-stationary data due to the fact that these two approaches require data to be stationary and generated by linear systems. To surmount this issue, several decomposition techniques have been proposed for analyzing non-linear and non-stationary time series. The leading rule of all these approaches is common: first, the signal is decomposed into simpler components, and second, time-frequency analysis is applied to each component separately. Two ways characterize the decomposition of a signal: either by iteration or optimization.

The earliest iterative algorithms for signal decomposition is the Empirical Mode Decomposition (EMD) [71], developed by Norden Huang in the late 1990s. EMD is a data-driven, non-parametric method that decomposes a signal into a finite number of intrinsic mode functions (IMFs) using an iterative sifting process.

The IMFs are defined as a set of oscillator functions fulfilling three properties: the number of extrema and zero crossings must be equal or differ by at most one; at any point in the signal, the mean value of the envelope defined by the local maxima and the envelope defined by the local minima must be zero; the waveform defined by the IMF should be symmetrical around the mean value of the envelope [71].

The EMD algorithm works by first identifying the extrema (maxima and minima) of the signal and connecting them with cubic splines to form upper and lower envelopes. The mean of these envelopes is then subtracted from the original signal to obtain the approximation of the first IMF, which represents the high-frequency component of the signal. The computation of the envelopes is reapplied to this first approximation of the first IMF and a new moving average is computed and subtracted from it. This procedure is iterated until a stopping criterion is fulfilled. This process is repeated on the residual signal (the original signal minus the first IMF) to obtain the second IMF, and so on until a stopping criterion is met.

The iterative structure of the EMD is described as follows: consider an operator \mathcal{O} performing the moving average of a signal $S(x)$, and an operator \mathcal{G} getting the fluctuation part, $\mathcal{G}(S)(x) = S(x) - \mathcal{O}(S)(x)$. We obtain the first IMF from the sifting process

$$I_1(x) = \lim_{n \rightarrow \infty} \mathcal{G}_{1,n}(S_n)(x) \quad (2.1)$$

we denote $S_n(x) = \mathcal{G}_{1,n-1}(S_{n-1})(x)$ and $S_1 = S(x)$. The limit ensure that the signal remain

the same when applied \mathcal{G} one more time.

By iterative sifting process, we obtain the remaining IMFs as follows:

$$I_k(x) = \lim_{n \rightarrow \infty} \mathcal{G}_{k,n}(r_n)(x), \quad (2.2)$$

with $r_n = \mathcal{G}_{k,n}(r_{n-1})(x)$ and $r_1(x) = r(x)$ which is the remainder $S(x) - I_1(x) - \dots - I_{k-1}(x)$. The sifting process stop when $r(x) = S(x) - I_1(x) - \dots - I_m(x)$ has at most one local maximum or minimum. It is easy to reconstruct the signal $S(x)$ from its IMFs and the remainder by

$$S(x) = \sum_{j=1}^m I_j(x) + r(x). \quad (2.3)$$

The operator \mathcal{O} is obtained as the mean function of the upper and lower envelope, where one identifies first the extrema (maxima and minima) of the signal $S(x)$ and then connects them with cubic splines to form an upper and lower envelope.

The stopping criterion is typically based on the amplitude of the last IMF relative to the noise level of the residual signal. Meaning that the algorithm stops when the last IMF become a trend that is further iteration has at most one local maximum or minimum [79]. Once the IMFs are obtained, they can be reconstructed by simply summing them up in order.

The EMD algorithm has been successfully applied to a wide range of signal processing tasks, including signal denoising, trend analysis, and feature extraction. However, it does have some limitations, such as sensitivity to noise and the potential for mode mixing (i.e., the presence of multiple frequencies in a single IMF). Several variants and extensions of the EMD algorithm have been proposed to address these issues.

An obvious perturbation problem is generated from the fact of repeatedly applying the cubic spline in each iteration, leading to an unstable method under perturbation. As a solution to the EMD issue, Huang et al. proposed the ensemble Empirical Mode Decomposition (EEMD) [72]. The main idea behind EEMD is to add noise to the original signal and then decompose the resulting noisy signal into IMFs using EMD. This process is repeated many times with different realizations of noise, and the resulting IMFs are averaged to obtain a more robust decomposition.

EEMD has several advantages over EMD, including the ability to handle signals with nonstationary and non-periodic components and the ability to reduce mode mixing, which occurs when different modes of a signal are mixed together in a single IMF. EEMD is also less susceptible to edge effects than EMD, which can lead to spurious modes at the beginning and end of the signal.

Another type of iterative decomposition is Iterative Filtering (IF), inspired by EMD [73]. The iterative filtering technique works by applying a filter to an image or signal, then subtracting the filtered version from the original. This difference is then added back to the original signal, and the process is repeated multiple times.

Each iteration helps to remove more noise or artifacts from the signal, resulting in a cleaner and clearer output. Iterative filtering is commonly used in applications such as image denoising, image restoration, and signal processing.

IF is based on the same framework as EMD, but the operator performing the moving average of a signal $S(x)$ is obtained by convolving $S(x)$ with low-pass filters, instead of computing the average of the upper and lower envelop as in EMD, thus creating the IMFs.

Let's consider a signal $S(x)$, $x \in \mathbb{R}$, and $\mathcal{O}(S(x))$ as a moving average of the signal $S(x)$, where \mathcal{O} is an operator. Let $L(t)$ be a low pass filter (the double average filter) defined for instance by

$$L(t) = \frac{l+1-|t|}{(l+1)^2}, \quad t \in [-l, l] \quad (2.4)$$

we therefore define the moving average of the signal $S(x)$ as the convolution

$$\mathcal{O}(S(x)) = \int_{-l}^l S(x+t)L(t)dt. \quad (2.5)$$

More in general, $\mathcal{O}(S(x))$, can be defined as a convolution of the signal $S(x)$ and some filter ω in the IF method. A function $\omega : [-l, l] \rightarrow \mathbb{R}$ is called a filter if it is nonnegative i.e. $\omega(t) > 0$, even i.e. $\omega(-t) = \omega(t)$, bounded i.e. $\exists 0 < M \in \mathbb{R}$ such that $|\omega(t)| \leq M, \forall t \in [-l, l]$, continuous, and $\int_{\mathbb{R}} \omega(t)dt = 1$ [83].

Let define $S_1(x) = S(x)$ and

$$\mathcal{G}_{1,n}(S_n) = S_n - \mathcal{O}_n^1(S_n) = S_{n+1}, \quad (2.6)$$

the operator capturing the fluctuation part of S_n for $n = 1, 2, \dots$, one obtain the first IMF as $I_1 = \lim_{n \rightarrow \infty} \mathcal{G}_{1,n}(S_n)$, here the operator \mathcal{O}_n^1 is linked to the mask length l_n which is the length of the filter during step n , the superscript 1 refer to the first IMF.

The mask length of iterative filtering is typically determined by the size of the kernel or filter used in the filtering process. The mask length refers to the number of elements in the filter that are used to perform the filtering operation. The mask length can be computed based on the specific requirements of the iterative filtering operation, such as the level of smoothing or detail preservation needed in the output image.

Following [73], the mask length is computed as

$$l_n = \left\lfloor \nu \frac{N}{k} \right\rfloor \quad (2.7)$$

Where N represents the total number of sample points of the signal $S_n(x)$, k represents the number of its extremum points, ν is a parameter fixed around 1.6, and $\lfloor \cdot \rfloor$ rounds a positive number to the nearest integer part close to zero [79].

In a similar way we obtain the second IMF I_2 by applying the operator \mathcal{G} to the remaining signal $S - I_1$. We therefore iterate the process to obtain the k -th IMF as $I_k = \lim_{n \rightarrow \infty} \mathcal{G}_{k,n}(r_n) = r_{n+1}$, where $r_1 = S - I_1 - \dots - I_{k-1}$. We stop the IF when $r = S - I_1 - \dots - I_m$, $m \in \mathbb{N}$, becomes

a trend signal which means that further iteration has at most one local maximum or minimum [79].

In practice, several conditions can be considered. Firstly, a maximum number of iterations should be predetermined in advance, and the iterative process is stopped once this number of iterations is reached. Another way to determine the stopping point is to consider a signal-to-noise ratio (SNR) improvement. In this case, the iterative process is stopped when the SNR of the filtered signal reaches a predefined level. Alternatively, visual inspection can be employed, meaning that the iterative process is stopped when the filtered signal visually meets the desired filtering goals.

Given a signal $S(x)$, $x \in \mathbb{R}$ the IF algorithm is performed by applying two nested loops: the inner one performs the necessary operations to obtain an IMF, and the outer one computes all the IMFs [79].

IF algorithm IMF=IF(S)

IMF = {}

while the number of extrema of $S \geq 2$ do

$S_1 = S$

while the stopping criterion is not satisfied do

compute the first length l_n for S_n

$S_{n+1}(x) = S_n(x) - \int_{-l_n}^{l_n} S_{n+1}(x+t)\omega_n(t)dt$

$n = n + 1$

end while

IMF = IMF \cup { S_n }

$S = S - S_n$

end while

IMF = IMF \cup { S }

In [79], to implement the IF algorithm, the mask length is only computed in the first step, and then the same value is used for all the remaining steps. There is a reason for doing so: to ensure that each IMF produced by the framework contains a well-defined set of instantaneous frequencies [79]. By following this idea, one can observe that the operators \mathcal{G} and \mathcal{O} do not depend on the step number n . Therefore, for a given signal $S(x)$, where $x \in \mathbb{R}$, the first IMF is obtained simply by $I_1 = \lim_{n \rightarrow \infty} \mathcal{G}^n(S)$, with $\mathcal{G}(S) = S - \mathcal{O}(S)$ and $\mathcal{O}(S) = \int_{-l}^l S(x+t)\omega(t)dt$, where l is the mask length computed only in the first step of the inner loop, and $\omega(t)$ is a convenient filter function. In the inner loop of the IF algorithm, the convergence is guaranteed for a periodic signal, and this convergence has been studied in the space of functions l^∞ in [84].

Let's explore this idea in depth. Consider a continuous signal $S(x) \in \mathbb{R}$, and a uniform filter $\omega(t)$, compactly supported on $t \in [-l, l]$. The operator \mathcal{O} is computed as follow

$$\mathcal{O}(S)(x) = \int_{-l}^l S(x+t)\omega(t)dt, \quad (2.8)$$

we can define the operator \mathcal{G} as

$$\mathcal{G}(S) = S - \mathcal{O}(S) = (1 - \mathcal{O})(S). \quad (2.9)$$

The first step of the IF algorithm is the application of the operator \mathcal{G} to the current signal. Furthermore, by fixing the mask length l throughout all the steps of an inner loop, we obtain a function sequence $\mathcal{G}^n(S)$. We summarize this idea in the following proposition to obtain a nice form of the sequence $\{\mathcal{G}^n(S)\}$ in order to think about its convergence.

Proposition 2.1. *Let l be the fixed mask length throughout all the steps of an inner loop, then the following equality hold*

$$\mathcal{G}^n(S) = (1 - O)^n(S). \quad (2.10)$$

Proof. Recall our first characterization of the $n + 1$ terms obtained by sifting the operator \mathcal{G} n times, i.e

$$S_{n+1} = \mathcal{G}_n(S_n) = \mathcal{G}^n(S) = (S - O(S))^n \text{ and } S_1 = S. \quad (2.11)$$

Let us prove the proposition by recurrence. For $n = 1$, $S_2 = (S - O(S))^1 = (1 - O)(S)$. For $n = 2$ and using the equation (2.6) we have

$$\begin{aligned} S_3 &= \mathcal{G}_2(S_2) = S_2 - O(S_2) = (1 - O)S_2 \\ &= (1 - O)(1 - O)S_1 \\ &= (1 - O)^2S_1, \end{aligned} \quad (2.12)$$

from (2.11) and (2.12) $S_3 = \mathcal{G}_2(S_2) = (1 - O)^2(S)$. Let us suppose that the (2.10) it is true for $n = k - 1$ i.e. $S_{k-1} = (1 - O)^{k-2}$ and now let us show that it remain true for $n = k$

$$S_k = \mathcal{G}_{k-1}(S_{k-1}) = \mathcal{G}^{k-1}(S_{k-1}) = (S_{k-1} - O(S_{k-1})) = (1 - O)S_{k-1} = (1 - O)^{k-1}S_1, \quad (2.13)$$

So from (2.11) and (2.13) we have $S_{k+1} = \mathcal{G}_k = \mathcal{G}^k = (1 - O)^kS_1$. \square

If we assume the sequence $\{\mathcal{G}^n(S)\}$ is convergent then we compute the first IMF as

$$I_1 = \lim_{n \rightarrow \infty} \mathcal{G}^n(S). \quad (2.14)$$

In [84] the authors provides a proof on the convergence of the sequence $\mathcal{G}^n(S)$. In [79] a proof of the sequence $\mathcal{G}^n(S)$ is provided when the signal S is in L^2 . The idea developed in [79] is to take a continuous filter $\omega(t)$ that is symmetric and compactly supported. By symmetry, we mean $\omega(-t) = \omega(t)$ for $t \in [-l, l]$. Such ω satisfying all the mentioned properties implies that $\omega(t) \in L^2(\mathbb{R})$. It now appears clearly that (2.8) is the convolution of the signal S and the filter ω :

$$\begin{aligned} O(S)(x) &= \int_{-l}^l S(x+t)\omega(t)dt = \int_{-l}^l S(x-t)\omega(t)dt \\ &= \int_{-\infty}^{\infty} S(x+t)\omega(t)dt = (S * \omega)(x). \end{aligned} \quad (2.15)$$

Since $\omega(t) \in L^2(\mathbb{R})$, we compute its Fourier transform as $\mathcal{F}(\omega)(\xi) = \int_{-\infty}^{\infty} \omega(t)e^{-2\pi i t \xi} dt$, $\xi \in \mathbb{R}$. We can apply the convolution theorem of the Fourier transform, to obtain $\mathcal{F}(O(S))(\xi) = \mathcal{F}(S * \omega)(\xi) = \mathcal{F}(S)(\xi)\mathcal{F}(\omega)(\xi)$, $\xi \in \mathbb{R}$. The ensuing proposition provide a nice form of the the Fourier transform of the equation (2.10).

Proposition 2.2. Let \mathcal{G} be the sifting operator of the signal S satisfying the equation (2.10), then the Fourier transform of the (2.10) is given by

$$\mathcal{F}(\mathcal{G}^n(S))(\xi) = \mathcal{F}((I - O)^n S)(\xi) = [1 - \mathcal{F}(\omega)(\xi)]^n \mathcal{F}(S)(\xi), \quad \xi \in \mathbb{R}. \quad (2.16)$$

Proof. We can apply the equation (2.10), (2.15) and use the linearity and convolutional property of the Fourier transform to obtain

$$\begin{aligned} \mathcal{F}(\mathcal{G}^n(S))(\xi) &= \mathcal{F}((S - O(S))^n)(\xi) = \mathcal{F}((S - \omega * S)^n)(\xi) = \mathcal{F}((S - \omega * S) \cdots (S - \omega * S))(\xi) \\ &= \mathcal{F}((\delta - \omega) * \cdots * (\delta - \omega) * S)(\xi) \\ &= \mathcal{F}(\delta - \omega)(\xi) \cdots \mathcal{F}(\delta - \omega)(\xi) \mathcal{F}(S)(\xi) \\ &= [1 - \mathcal{F}(\omega)(\xi)] \cdots [1 - \mathcal{F}(\omega)(\xi)] \mathcal{F}(S)(\xi) \\ &= [1 - \mathcal{F}(\omega)(\xi)]^n \mathcal{F}(S)(\xi) \end{aligned} \quad (2.17)$$

□

We have the necessary ingredient to present the convergence theorem of the sequence $\{\mathcal{G}^n(S)\}$.

Theorem 2.3. Consider a symmetric non negative filter $\omega(t) \in L^2(\mathbb{R})$, $t \in [-l, l]$ with $\int_{-l}^l \omega(t) dt = 1$, and let the signal $S(x)$ be square integrable i.e $S(x) \in L^2(\mathbb{R})$. If $|1 - \mathcal{F}(\omega)(\xi)| < 1$ or $\mathcal{F}(\omega)(\xi) = 0$. Then the sequence $\{\mathcal{G}^n(S)\}$ converges and

$$\lim_{n \rightarrow \infty} \mathcal{G}^n(S)(x) = \int_{-\infty}^{\infty} \mathcal{F}(S)(\xi) \chi_{\{\mathcal{F}(\omega)(\xi)=0\}} e^{2\pi i \xi x} d\xi \quad (2.18)$$

Proof. For $S(x)$ $x \in \mathbb{R}$ we can apply a result from Plancherel theorem stating that the integral of the square of the Fourier transform of a function is equal to the integral of the square of the function itself, i.e

$$\int_{-\infty}^{\infty} |\mathcal{F}(S)(\xi)|^2 d\xi = \int_{-\infty}^{\infty} |S(x)|^2 dx < \infty. \quad (2.19)$$

We have two cases to take into account, either $|1 - \mathcal{F}(\omega)(\xi)| < 1$ or $\mathcal{F}(\omega)(\xi) = 0$.

Let us consider the case $|1 - \mathcal{F}(\omega)(\xi)| < 1$

$$\begin{aligned} |\mathcal{F}(\mathcal{G}^n(S))(\xi)| &= |[1 - \mathcal{F}(\omega)(\xi)]^n \mathcal{F}(S)(\xi)| = |1 - \mathcal{F}(\omega)(\xi)|^n |\mathcal{F}(S)(\xi)| \\ &< |\mathcal{F}(S)(\xi)|, \end{aligned} \quad (2.20)$$

we now possess two nice information to infer on the convergence of the sequence $\mathcal{F}(\mathcal{G}^n(S))(\xi)$, which is: $|1 - \mathcal{F}(\omega)(\xi)|^n$ is a geometric sequence and S is in L^2 , so we have

$$\lim_{n \rightarrow \infty} |\mathcal{F}(\mathcal{G}^n(S))(\xi)| = 0, \quad (2.21)$$

by the property of convergent sequence $\lim_{n \rightarrow \infty} \mathcal{F}(\mathcal{G}^n(S))(\xi) = 0$.

Now for the case $\mathcal{F}(\omega)(\xi) = 0$, we have

$$|\mathcal{F}(\mathcal{G}^n(S))(\xi)| = |1 - \mathcal{F}(\omega)(\xi)|^n |\mathcal{F}(S)(\xi)| = \mathcal{F}(S)(\xi). \quad (2.22)$$

Combining the two cases we have

$$\lim_{n \rightarrow \infty} \mathcal{F}(\mathcal{G}^n(S))(\xi) = \begin{cases} 0 & \text{for } |1 - \mathcal{F}(\omega)(\xi)| < 1, \\ \mathcal{F}(S)(\xi) & \text{for } \mathcal{F}(\omega)(\xi) = 0. \end{cases} \quad (2.23)$$

From the fact that Fourier transform is an invertible operator, the sequence $\{\mathcal{G}^n(S)\}$ is also convergent as an inverse Fourier transform of the convergent sequence $\{\mathcal{F}(\mathcal{G}^n(S))(\xi)\}$ and admit as limit

$$\lim_{n \rightarrow \infty} \mathcal{G}^n(S)(x) = \int_{-\infty}^{\infty} \mathcal{F}(S)(\xi) \chi_{\{\xi: \mathcal{F}(\omega)(\xi)=0\}} e^{2\pi i \xi x} d\xi \quad (2.24)$$

□

This theorem is of great importance, since it provides sufficient conditions on the filter that guaranty the convergence of the inner loop of the IF algorithm. We emphasize that (2.24) is actually an explicit formula for the IMF of a signal S based on IF algorithm with a filter ω . Recall that our previous assumption on the sufficient conditions on the filter i.e. assuming it to be a low pass filter, for example the double average filter $L(t)$ is not unrealistic. In fact $L(t)$ satisfies the condition of the above theorem if $\xi = \frac{k}{l+1}$ $1 \leq k \leq l+1$ under this requirement $\mathcal{F}(L)(\xi) = 0$. By exploring the world of kernel we realize that filters satisfying properties $|1 - \mathcal{F}(\omega)(\xi)| < 1$ and $\mathcal{F}(\omega)(\xi) = 0$ are not unique and can be easily found. To explore a little bit this world, let us consider a symmetric and non negative filter ω , with $\int_{-l}^l \omega(t) dt = 1$ we can compute its Fourier transform

$$\mathcal{F}(\omega)(\xi) = \int_{-\infty}^{\infty} \omega(t) \cos(-2\pi i t \xi) dt. \quad (2.25)$$

The modulus of (2.25) equals

$$\begin{aligned} |\mathcal{F}(\omega)(\xi)| &= \left| \int_{-\infty}^{\infty} \omega(t) \cos(-2\pi i t \xi) dt \right| \leq \int_{-\infty}^{\infty} |\omega(t) \cos(-2\pi i t \xi)| dt \\ &< \int_{-\infty}^{\infty} |\omega(t)| dt = \int_{-l}^l \omega(t) dt = 1. \end{aligned} \quad (2.26)$$

From (2.26) we infer that for a non-negative and symmetric filter $\omega(t)$, $t \in [-l, l]$, $-1 < \mathcal{F}(\omega)(\xi) < 1$, for every $\xi \in \mathbb{R}$. For $\omega(t)$ such that $|1 - \mathcal{F}(\omega)(\xi)| < 1$ we need a particular $\omega(t)$ such that $0 \leq \mathcal{F}(\omega)(\xi) < 1$, to have such a condition we can chose a filter $u(t)$, $t \in [-2l, 2l]$, obtained as a convolution of a symmetric non-negative filter $\omega(t)$, $t \in [-l, l]$ with itself, so we have

$$u(t) = \omega(t) * \omega(t). \quad (2.27)$$

We obtain the expected result by computing the Fourier transform of $u(t)$, which is $\mathcal{F}(u)(\xi) = \mathcal{F}(\omega)(\xi) \cdot \mathcal{F}(\omega)(\xi)$, and it is now clear that $0 \leq \mathcal{F}(u)(\xi) < 1$, $\forall \xi \in \mathbb{R}$. Therefore, any filter given as convolution of a symmetric, non-negative and compactly supported filter in L^2 space with itself

satisfies the conditions of Theorem 2.3.

Let's consider the artificial signal $S(t) = \sin(4\pi t) + 0.5 \cos(5\pi|t| - 40\pi t^2)$, depicted in Figure 2.1. This signal is a sum of two components: one with amplitude 1 and zero phase, and the other with an amplitude of 0.5. After applying the IF algorithm, the signal reveals three IIMFs, as shown in Figure 2.2.

The first IMF, c_1 corresponds to the signal component $0.5 \cos(5\pi|t| - 40\pi t^2)$, the second IMF, c_2 capture the portion of the signal $\sin(4\pi t)$, and the third IMF, c_3 , represent the mean of the original signal.

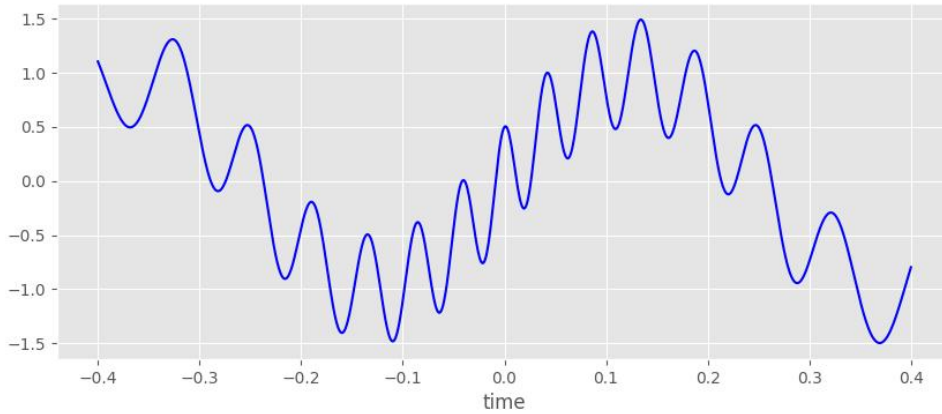


Figure 2.1: Example of the artificial signal $S(t) = \sin(4\pi t) + 0.5 \cos(5\pi|t| - 40\pi t^2)$

The Iterative filtering has also its generalizations. Among them there is the so called adaptive local iterative filter (ALIF) [79]. The ALIF algorithm is based on the IF method. The working difference are just the way the filter mask is computed.

It is of great importance to study the convergence of the inner loop of the IF for obtaining each IMF. IF is stable under perturbation and the convergence is guaranteed for periodic and l^∞ signals using uniform filters. However, the convergence for general signals with uniform and non-uniform filters cannot be proven under IF. The ALIF technique relies on computing the mask length $l_n(x)$. This mask length is required to be a positive function, giving rise to two perspectives:

1. The mask length is a positive constant function, essentially transforming the ALIF technique into the IF method.
2. The mask length varies from point to point, resulting in a non-uniform mask length.

The convergence of the ALIF method is assured in the case of a constant mask length, as stated in Theorem 2.3. However, for the convergence of the ALIF method with a non-uniform mask, it is necessary to remove the high-frequency oscillations present in the mask length $l_n(x)$. Therefore, convergence is guaranteed when dealing with a slowly varying mask length [79].

The ALIF algorithm is the same as IF, the only change is in the way the operator capturing the fluctuation part is computed. This operator is given for a signal $S(x)$ $x \in \mathbb{R}$ by

$$\mathcal{G}_{1,n}(S_n) = S_n - \mathcal{O}_{\omega, l_n}^1(S_n) = S_{n+1}, \quad (2.28)$$

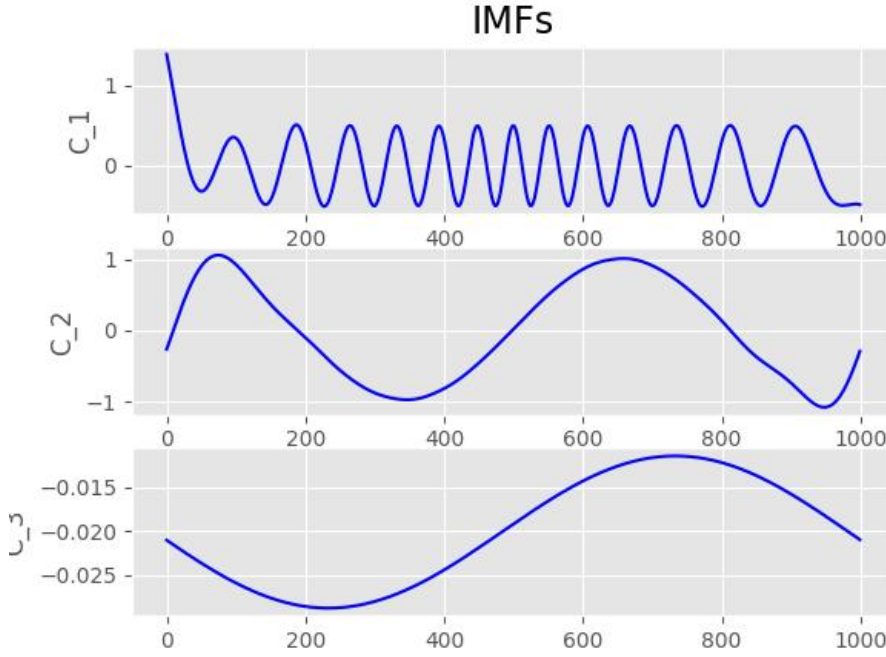


Figure 2.2: The IMFs of the artificial signal $S(t)$

and the moving average operator is given by

$$O_{\omega, l_n}^1(S_n) = \int_{-l_n(x)}^{l_n(x)} S_n(x+t) \omega_n^{(x)}(t) dt \quad (2.29)$$

The first IMF is computed in a similar way as in IF i.e $I_1 = \lim_{n \rightarrow \infty} \mathcal{G}_{1,n}(S_n)$, the operator $\mathcal{G}_{1,n}$ depends on the mask length $l_n(x)$ computed at step n . The mask length $l_n(x)$ is a crucial aspect of the ALIF process. It must be a strictly positive function. When it remains constant for every x , it results in a uniform mask length, rendering the ALIF method equivalent to IF. On the other hand, when the mask length $l_n(x)$ varies from point to point, it generates a non-uniform mask length.

We can remind that the process applied to obtain the first IMF is a kind of sift for separating the finest local mode from the data. It is also natural that in practice we need a stopping criterion since we cannot let n tends to infinity in a computer for example. We first iteratively compute the quantity $I_{1,n} = \mathcal{G}_{1,n}(S_n)$, where at step n of the first inner loop $\mathcal{G}_{1,n}$ stands for an operator capturing the fluctuation part. To set a stopping criterion we use the standard deviation

$$SD := \frac{\|I_n - I_{n-1}\|_2}{\|I_{n-1}\|_2}. \quad (2.30)$$

In [71] the authors suggest to set this value between $[0.2, 0.3]$. Whereas in [81] SD is set in the interval $[0.001, 0.2]$. Different stopping criteria can also be considered for each inner loops. To obtain all the IMFs with ALIF, we proceed in the same way as we do for IF, by applying the previous sifting process to the remainder signal $r = S - I_1 - \dots - I_{k-1}$ $k \in \mathbb{N}$. The algorithm stop when the remainder signal r remain with at most one local extremum.

We can notice at this point the difference between the ALIF and the IF reside on the mask length $l_n(x)$ depending on x to perform the moving average using the operator \mathcal{O} . The mask length $l_n(x)$ has to be a strictly positive function. However, it can also be a strictly positive constant and in that case the ALIF reduces to IF framework. We can notice that the choice of the mask length $l_n(x)$ is not unique.

One of the crucial problem we cannot ignore since we adopt the same idea for computing the IMFs is that of the convergence of the ALIF. It is worth to notice that in the ALIF we are concerned with a non-uniform mask length, we need a slowly varying mask length. We have pointed out that when the mask length $l_n(x)$ is constant the ALIF reduces to IF and if in a limit case $l_n(x)$ does not change at all, so the ALIF reduces to IF and without disquiet we apply Theorem 2.3.

Let consider the ALIF algorithm with a non-uniform mask length and let consider the operator

$$\mathcal{O}_{\omega, l}(S) = \int_{-l(x)}^{l(x)} S(x+t)\omega^{(x)}(t)dt, \quad (2.31)$$

and consider the sifting operator $\mathcal{G}_n(S_n)$ given by

$$S_{n+1} = \mathcal{G}_n(S_n)(x) = S_n(x) - \int_{-L}^L S_n(x + g_n(x, y))W(y)dy, \quad (2.32)$$

we precise that $W(y), \in [-L, L]$ is a filter and $g_n(x, y)$ is a scaling function defined on $\mathbb{R} \times [-L, L]$ having value on $\mathbb{R} \times [-l_n(x), l_n(x)]$, and can be seen as a linear function $g_n(x, y) = l_n(x)y/L$ or it also can be regarded as cubic function $g_n(x, y) = l_n(x)y^3/L^3$

Theorem 2.4. *Let consider a continuous signal $S(x)$ $x \in \mathbb{R}$ in $L^\infty(\mathbb{R})$. Let consider the sequence*

$$\epsilon_n = \frac{\|\mathcal{O}_{\omega_{n+1}, l_{n+1}}(S_{n+1})\|_{L^\infty}}{\|\mathcal{O}_{\omega_n, l_n}(S_n)\|_{L^\infty}}, \quad \delta_n = \frac{\|\mathcal{O}_{\omega_{n+1}, l_{n+1}}(|S_{n+1}|)\|_{L^\infty}}{\|\mathcal{O}_{\omega_n, l_n}(|S_n|)\|_{L^\infty}}. \quad (2.33)$$

If

$$\prod_{i=1}^n \epsilon_i \rightarrow 0, \quad \prod_{i=1}^n \sigma_i \rightarrow c > 0, \quad \text{as } n \rightarrow \infty. \quad (2.34)$$

Then $\{S_n(x)\}$ converge almost every where to an IMF.

It is worthwhile to notice once more that even the mask length $l_n(x)$ inside the inner loop can be computed at each step n , we only compute it in the first step in the implemented code and use that mask length for following steps. This way of doing lead the operators \mathcal{G} and \mathcal{O} to be independent on the steps n , which is to say, given a signal $S(x)$ $x \in \mathbb{R}$ the operators are given by $\mathcal{G}(h) = h - \mathcal{O}(h)$, and $\mathcal{O}(h)(x) = \int_{-l(x)}^{l(x)} h(x+t)\omega^{(x)}(t)dt$, where $l(x)$ is the mask length computed in the first step of the sifting process, $h = S - I_1 \cdots I_{k-1}$, and $\omega^{(x)}(t)$ a convenient filter with compact support in the interval $[-l(x), l(x)]$.

In this framework we generate the moving average of a given signal by making use of the FP filters obtained as a solution of Fokker-Plank equations. Another way of computing filters is to use in fact partial differential equations (PDEs). This is because PDEs are used to model diffusion

processes. Therefore given a diffusion PDEs we can make use of it fundamental solution to build a filter for the ALIF or IF framework. A better candidate for generating a smooth compactly supported filters is the state-of-the-art Fokker-Plank equation and we call these filter FP filters. Let be given a Fokker-Plank equation

$$p_t = -\alpha (f(x)p)_x + \beta (g^2(x)p)_{xx}, \quad \alpha, \beta > 0, \quad (2.35)$$

where the function $h(x)$ and $g(x)$ are smooth enough and for $a < 0 < b$ the following conditions are satisfied:

$$\bullet \quad g(a) = g(b), \quad g(x) > 0 \text{ for } x \in (a, b) \quad (2.36)$$

$$\bullet \quad h(a) < 0 < h(b) \quad (2.37)$$

There exist a non-trivial smooth solution of the homogeneous problem

$$-\alpha (f(x)p)_x + \beta (g^2(x)p)_{xx} = 0, \quad (2.38)$$

where $p(x)$ satisfy the two conditions $p(x) \geq 0$ for $x \in (a, b)$ and $p(x) = 0$ for $x \notin (a, b)$.

2.2 IMFogram

The time-frequency representation of the intrinsic mode functions (IMFs) appears to be a challenging task, as each IMF lacks fixed frequencies and is not orthogonal to one another. This differs from the periodogram and its localized counterpart, the spectrogram, which are indeed graphical representations tailored for Fourier analysis. Since each Fourier component possesses a distinct frequency, and these components are independent of each other, the total energy of the signal results from the sum of the energy of each component. The IMFogram is a straightforward analog of the spectrogram, designed for rapid computation based on IMF decompositions. Its definition is as follows:

Let consider η as a parameter between 5 to 20 in practice, and let denote by f an IMF produced with a filter length l . The approximation of its local energy at t is given as follow:

$$E_f(t) = \frac{1}{2\eta l} \int_{t-\eta l}^{t+\eta l} f(\tau)^2 d\tau. \quad (2.39)$$

The fact that an IMF focuses on a narrow frequency section, this fact provide an intuition on an approximation of f by

$$\Omega_f(t) = \frac{1}{4\eta l} \times \text{number of zero-crossings of } f \text{ over the interval } [t - \eta l, t + \eta l]. \quad (2.40)$$

It is worth noting that the approximation described in (2.40) is not unique. There are many other very well highlighted in [84][82]. The above local energy and frequency approximation of f provide the frequency, time, and energy triples, $(t_i, \Omega_f(t_i), E_f(t_i))_{i \in \mathbb{Z}_N}$ for f [82].

Let consider a signal α defined on $[0, L]$ sampled at rate B per time unit. We represent it as vector of size $N = BL$. We wrap it on a discrete circle: consider a quotient group $\mathbb{Z}_N = \mathbb{Z}/N\mathbb{Z}$ and set $t_i = i/B$, $i \in \mathbb{Z}_N$. The discretized and wrapped signal is $s = (\alpha(t_i))_{i \in \mathbb{Z}_N}$ with time domain

$\{i/B : i \in \mathbb{Z}_N\}$ and frequency domain $(\mathbb{Z}_N/L) \cap [0, B/2]$. Let partition the time-frequency domain in rectangles in order to obtain a time-frequency representation of the signal. For a given rectangle R , we denote by $\Pi_t R$ the projection of the rectangle onto the time coordinate and $\Pi_\omega R$ the projection onto the frequency coordinate. Let s_1, s_2, \dots, s_k be the IMF decomposition of s . We define an energy associated to a rectangle R as a sum of the average local energies of each IMF when the local frequency lies in Π_ω , i.e

$$E_s(R) = \sum_{1 \leq j \leq k} \frac{1}{\#\Pi_t R} \sum_{\tau \in \Pi_t R} E_{s_j}(\tau) \mathbb{1}\{\Omega_f(\tau) \in \Pi_\omega R\} \quad (2.41)$$

It is important to choose the length of $\Pi_t R$ to be comparable to the smallest filter length $l(\omega_1)$, in this case changes in high frequencies are well represented. The choice of the rectangle R to cover the time and frequency domain is subject to the type of application [82].

The IMFogram of the signal s is nothing else than the plot of the step function that equals to $E_s(R)$ on each rectangle R . The IMFogram of the artificial signal depicted in Figure 2.1 and its IMFs illustrated in the Figure 2.2 is depicted in the Figure 2.3 [82].

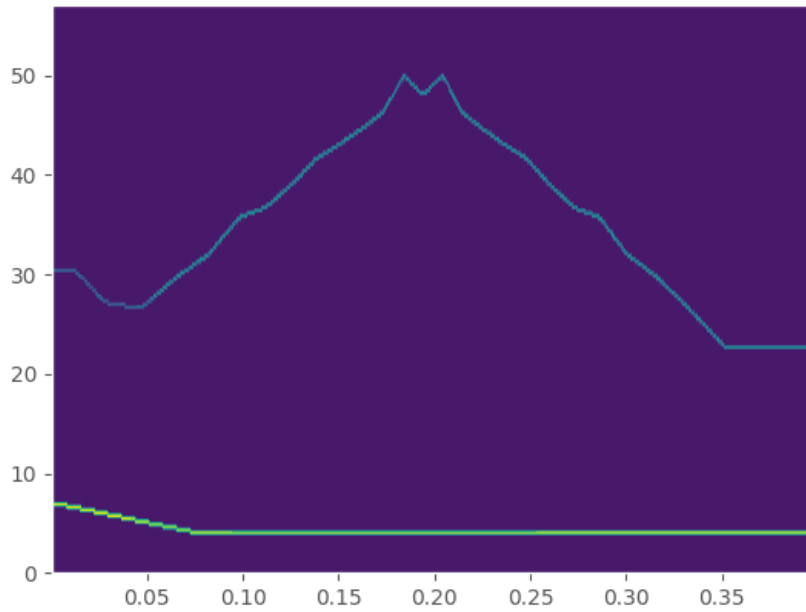


Figure 2.3: The IMFogram of the figure 2.1 and 2.2

3 Artificial neural network and Convolutional Neural Networks

3.1 Neural Network

A neural network (NN) is a collection of weights, together with the associated realization of the NN. A precise definition of a NN is given above by following the one presented in .

Definition 3.1. [85, 86] A neural networks Φ with input dimension d and L layers is a collection of matrix-vector tuples

$$\Phi = ((A_1, b_1), \dots, (A_L, b_L)), \quad (3.1)$$

where $N_0 = d, L \in \mathbb{N}$ and $N_1, \dots, N_L \in \mathbb{N}$ and each A_j are matrices of type (N_j, N_{j-1}) and b_j is vector of dimension N_j for $j = 1, \dots, L$. If Φ is neural network as described above, we define the realization of the NN Φ as a function defined from \mathbb{R}^{N_0} to \mathbb{R}^{N_L} , such that:

$$R_\Phi(\Phi) : \mathbb{R}^d \longrightarrow \mathbb{R}^{N_L} : x \longmapsto x_L =: R_\Phi(\Phi)(x),$$

it is worth noting that x_L is generated from the following design:

$$\begin{aligned} x_0 &:= x \\ x_j &:= \varrho(A_j x_{j-1} + b_j) \quad j = 1 \dots L-1 \\ x_L &:= A_L x_{L-1} + b_L, \end{aligned} \quad (3.2)$$

it is of great importance to emphasize that ϱ is a special function usually called an activation function, which can be of different types, e.g., ReLU, sigmoid, and tangent hyperbolic. This special function act component-wise on vector-valued input, i.e., $\varrho(z) = (\varrho(z_1), \dots, \varrho(z_L))$ for $z = (z_1, \dots, z_L) \in \mathbb{R}^L$. The number $N(\Phi) = d + \sum_{i=1}^L N_i$ is the total number of neurons in the NN Φ , $L(\Phi) := L$ is the number of layers, also referred to as the depth of NN, $M_j = \|A_j\|_0 + \|b_j\|_0$ is the total number of nonzero weights in the j -th layer, and finally $M(\Phi) = \sum_{j=1}^L M_j$ is the total number of nonzero weights in the NN Φ , also called the size of the NN. The number N_L represent the output dimension of the NN Φ [85, 86].

It is worthwhile to remark that if the product $A_j x_{j-1}$ in equation (3.2) is replaced by a convolutional or cross-correlation operation with some addition ingredient as described bellow, then the NN Φ is called a convolutional NN (CNN). The cross-correlation or the convolutional operation is described by

$$(A_j * x_j)_{t,p} = \sum_{i=1}^{N_j} \sum_{l=1}^{N_{j-1}} A_{j,i,l} x_{j,t+i,p+l}$$

Neural networks can also be constructed from existing ones by applying operations such as concatenation and parallelization. This is a way of building complex neural networks rather than just using simpler blocks.

Definition 3.2. Consider two natural numbers L_1, L_2 , and Φ^1, Φ^2 be two NNs of respective depth L_1 and L_2 with

$$\Phi^1 = ((A_1^1, b_1^1), \dots, (A_{L_1}^1, b_{L_1}^1)), \quad \Phi^2 = ((A_1^2, b_1^2), \dots, (A_{L_2}^2, b_{L_2}^2)),$$

Such that $N_0^1 = N_{L_2}^2 = d$, meaning that the input layer of the NN Φ^1 has the same dimension as the output layer of the NN Φ^2 . Then there exists an NN $\Phi^1 \bullet \Phi^2$ called the concatenation of Φ^1 and Φ^2 , which is defined as follows:

$$\Phi^1 \bullet \Phi^2 = ((A_1^2, b_1^2), \dots, (A_{L_2-1}^2, b_{L_2-1}^2), (A_1^1 A_{L_2}^2, A_1^1 b_{L_2}^2 + b_1^1), (A_2^1, b_2^1), \dots, (A_{L_1}^1, b_{L_1}^1)). \quad (3.3)$$

The concatenation $\Phi^1 \bullet \Phi^2$ has $L_1 + L_2 - 1$ layers, $R_\Phi(\Phi^1 \bullet \Phi^2) = R_\Phi(\Phi^1) \circ R_\Phi(\Phi^2)$ [85][86]

Proposition 3.3. For any natural numbers d and L , there exists an NN $\Phi_{d,L}^{\text{Id}}$ with L layers, where the total number of nonzero weights is less than or equal to $2dL$. Furthermore, the associated realization is simply the identity on \mathbb{R}^d , i.e., $R_\varrho(\Phi_{d,L}^{\text{Id}}) = \text{Id}_{\mathbb{R}^d}$. It is worth noting that this definition is only true if the activation function ϱ is of the ReLU type. Specifically, $\Phi_{d,L}^{\text{Id}}$ is defined as follows:

$$\Phi_{d,L}^{\text{Id}} = \left(\left(\begin{pmatrix} \text{Id}_{\mathbb{R}^d} \\ -\text{Id}_{\mathbb{R}^d} \end{pmatrix}, 0 \right), \underbrace{(\text{Id}_{\mathbb{R}^{2d}}, 0), \dots, (\text{Id}_{\mathbb{R}^{2d}}, 0)}_{L-2 \text{ times}}, ([\text{Id}_{\mathbb{R}^d} \mid -\text{Id}_{\mathbb{R}^d}], 0) \right). \quad (3.4)$$

We can now define a sparse concatenation [85, 86].

Definition 3.4. Let L_1, L_2 be in \mathbb{N} , $\varrho : \mathbb{R} \rightarrow \mathbb{R}$ be a ReLU, and let $\Phi^1 = ((A_1^1, b_1^1), \dots, (A_{L_1}^1, b_{L_1}^1))$ and $\Phi^2 = ((A_1^2, b_1^2), \dots, (A_{L_2}^2, b_{L_2}^2))$ be neural networks with the property that the input of the NN Φ^1 has the same dimension d as the output layer of the NN Φ^2 . Then there exist a NN $\Phi^1 \odot \Phi^2 := \Phi^1 \bullet \Phi^{\text{Id}} \bullet \Phi^2$ called the space concatenation of the NN Φ^1 and Φ^2 , and Φ^{Id} is defined as in Proposition 3.3, the sparse concatenation $\Phi^1 \odot \Phi^2$ has $L_1 + L_2$ layers, $R_\varrho(\Phi^1 \odot \Phi^2) = R_\varrho(\Phi^1) \circ R_\varrho(\Phi^2)$ and the total number on nonzero weights $M(\Phi^1 \odot \Phi^2) \leq 2M(\Phi^1) + 2M(\Phi^2)$. Another fundamental operation over NN is parallelization and one can construct it as follows [85, 86]

Proposition 3.5. Consider the natural numbers L and d , and let Φ^1 and Φ^2 be two NNs, each having L layers and a d -dimensional input. Then there exists an NN $P(\Phi^1, \Phi^2)$ with L layers and input dimension d , called the parallelization of Φ^1 and Φ^2 , defined as follows:

$$P(\Phi^1, \Phi^2) := ((\tilde{A}_1, \tilde{b}_1), \dots, (\tilde{A}_L, \tilde{b}_L)),$$

where

$$\tilde{A}_1 := \begin{pmatrix} A_1^1 \\ A_1^2 \end{pmatrix}, \quad \tilde{b}_1 := \begin{pmatrix} b_1^1 \\ b_1^2 \end{pmatrix}, \quad \text{and} \quad \tilde{A}_j := \begin{pmatrix} A_j^1 & 0 \\ 0 & A_j^2 \end{pmatrix}, \quad \tilde{b}_j := \begin{pmatrix} b_j^1 \\ b_j^2 \end{pmatrix}, \quad \text{for } 1 < j \leq L.$$

Where

$$R_\varrho(P(\Phi^1, \Phi^2))(x) = (R_\varrho(\Phi^1)(x), R_\varrho(\Phi^2)(x)) \quad \forall x \in \mathbb{R}^d$$

and $M(P(\Phi^1, \Phi^2)) = M(\Phi^1) + M(\Phi^2)$ [85, 86].

It is sometimes of great importance to use a parallelization that has two different inputs, and this scheme is defined as follows:

Proposition 3.6. Let $L \in \mathbb{N}$ and consider

$$\Phi^1 = ((A_1^1, b_1^1), \dots, (A_L^1, b_L^1)), \quad \text{and} \quad \Phi^2 = ((A_1^2, b_1^2), \dots, (A_L^2, b_L^2))$$

Let Φ^1 and Φ^2 be two NNs, both having L layers and inputs $N_0^1 = d_1$ and $N_0^2 = d_2$, respectively. Then there exists an NN $FP(\Phi^1, \Phi^2)$ called the full parallelization of Φ^1 and Φ^2 , possessing L layers and an input dimension of $d = d_1 + d_2$. For all $x = (x_1, x_2) \in \mathbb{R}^d$, where $x_i \in \mathbb{R}^{d_i}$ for $i = 1, 2$, it is defined as follows:

$$FP(\Phi^1, \Phi^2) := ((A_1^3, b_1^3), \dots, (A_L^3, b_L^3))$$

where, for $j = 1, \dots, L$ we set

$$A_j^3 := \begin{pmatrix} A_j^1 & 0 \\ 0 & A_j^2 \end{pmatrix} \quad b_j^3 := \begin{pmatrix} b_j^1 \\ b_j^2 \end{pmatrix}$$

We can now provide more details on the design of the convolutional neural network, which is a special case of the NN described above.

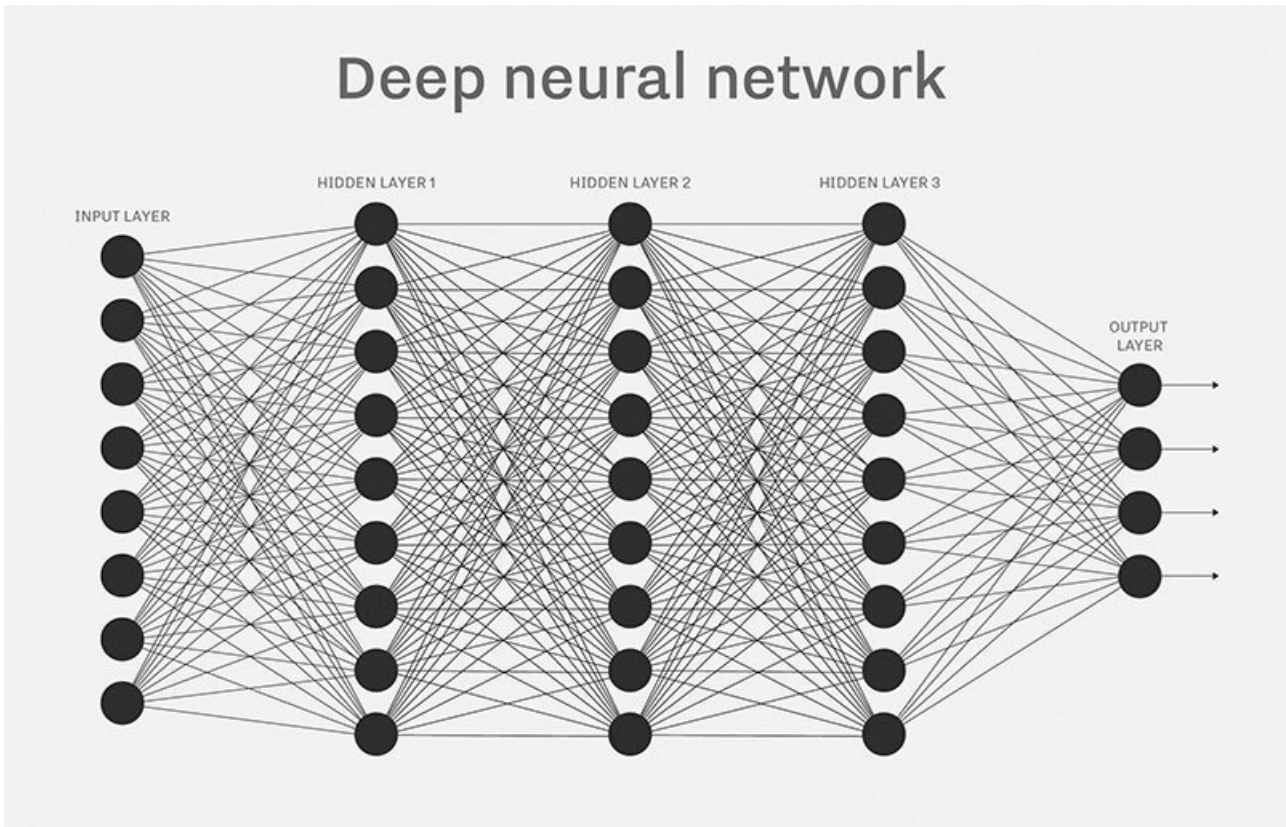


Figure 3.1: the artificial neural network ANN

3.2 Convolutional Neural Network

Convolutional neural networks (CNN), also called convolutional networks (ConvNet), are a type of neural network for processing data that have a grid topology. Typical examples include time series data defined as a one-dimensional grid at regular time intervals and image data defined as a two-dimensional pixel grid. The name convolutional neural networks is not accidental, it is used to refer to the use of the mathematical convolution operation in the network.

The idea of the CNN comes from a groundbreaking discovery of the visual cortex system by David Hubel and Torsten Wiesel. The CNN analyzes images by following the map of the visual cortex by adding convolution operations in some layers.

A Convolutional Neural Network (CNN) is composed of neurons that incorporate nonlinearity, weight parameters, biases, and a loss function to evaluate the overall errors of the system. The network employs backpropagation to rearrange its layers. CNN also uses multiple three-dimensional kernels that can slide through the input tensor like a window. The goal of sharing kernels is to reduce the number of parameters that the system needs to learn. Each layer's kernels extract features from the input tensor. The CNN architecture typically includes a Convolutional layer, a layer for nonlinear activation functions, a Batch Normalization layer (BN), a Pooling layer, a Dropout layer, and a Fully Connected layer (FC). We will briefly describe these layers below.

3.2.1 Convolutional Layer

A convolutional layer is a key building block of convolutional neural networks (CNNs) and is designed with the ultimate goal of extracting features from images or other spatial data. The layer contains a set of learnable filters (kernels) that convolve over the input data to generate a set of output feature maps. The size of the kernels are always smaller than the size of the input tensor otherwise the CNN is just a normal neural network [87].

Throughout the convolution operation, the filter passes over the input data, multiplying it element-by-element with the filter weights. The result of these multiplications is a single value that is then shown in the appropriate location on the output feature map.

The structures of the input data are conserved in the output feature maps which also comprises the needed information related to the spatial patterns, which are both feed to the subsequent layers of the CNN for multiple tasks comprising classification, object detection, and segmentation. An alternative way of creating deeper network for the ultimate purposes of learning complex representations of the input tensor is to stack convolutional layers on top of each other. Let us describe now the architecture of the convolutional layer.

As an essential compartment of the CNN, the convolutional layer takes as input a 3D, 2D or even 1D tensor data, and can have c channels of 2D or even 1D feature maps of size $h \times m$. The input is denoted by $X \in \mathbb{R}^{c \times h \times m}$. Each of the c feature maps convolves with a set of n kernels of size 3 i.e. each kernel is $K_i \in \mathbb{R}^{c \times s \times s}$ $1 \leq i \leq n$ to produce the output. The output obtained from the convolution operation is a 3D tensor Y of size $\tilde{h} \times \tilde{m}$ where $\tilde{h} = (h - s + 2p)/t + 1 = h - s + 1$ and $\tilde{m} = (m - s + 2p)/t + 1 = m - s + 1$. Here, p represents padding taken as zero, and t represents the stride taken as 1. The output Y is obtained by computing the convolution of the input data X_j , $1 \leq j \leq c$, with the kernel K_i , $1 \leq i \leq c$, so

$$Y_i = \sum_{j=1}^c K_{i,j} * X_j, \quad (3.5)$$

here $*$ stand for a 2D discrete convolutional operator, explicitly this operation is defined as

$$Y_{i,t,p} = \left(\sum_{j=1}^c K_{i,j} * X_j \right)_{t,p} = \sum_{j=1}^c \sum_{j_1=1}^s \sum_{j_2=1}^s K_{i,j,j_1,j_2} X_{j,t+j_1,p+j_2} \quad (3.6)$$

3.2.2 Non Linear Activation Layer

An essential component of the activation layer is the activation function which is a kind of master chief giving order to a neuron to activate or not. The activation function decide on the importance of the input to the network by applying simple mathematical functions. An activation function also called transfer function in artificial intelligence is a kind of mathematical gate separating the input data through the current neuron and its output moving to the next layer. Non-linear activation function allow the network to learn and extract complex feature from the input data. It models and approximate non-linear relationships in the input data. It also empower the network with the ability to solve a wide range of problems.

The non-linear activation function enables complex decision boundaries, meaning the network is now capable of capturing complex decision-making processes and classifying inputs that are not linearly separable. It is worth noticing that the choice of an activation function affects the quality of the given network to be expressive. This expressiveness empowers the network with the ability to learn and represent more involved features, ameliorating its ability to generalize and make accurate predictions on invisible data patterns.

The quality of the activation function eases the overall training ability of the network through backpropagation. The derivative of the activation function directly affects the magnitude of the gradient propagated through the learning process. We can use a specific activation function to alleviate the problem of vanishing or exploding gradients to ensure an efficiently stable learning process. Recent implementations prefer the use of non-linear activation functions such as Rectified Linear Unit (ReLU) [41] and PReLU [35].

3.2.3 Batch Normalization Layer

The distribution of the networks changes with respect to the learning process through each epoch. This change makes the training of the network cumbersome. To alleviate this issue one need to incorporate to the network batch normalization layer which will perform normalization for each training mini batch, which in turn will accelerate the convergence of the learning.

3.2.4 Pooling Layer

Pooling layers offer a method for downsampling feature maps by summarizing the presence of features within patches of the feature map. Two commonly used pooling techniques are average pooling, which provides an overview of the average presence of a feature, and max pooling, which highlights the most activated presence of a feature.

It is a common practice to incorporate a pooling layer after the convolutional layer, forming a recurring pattern in the layer arrangement of a convolutional neural network. This pooling layer independently processes each feature map, generating a new set of pooled feature maps with the same quantity.

The pooling operation entails selecting a specific operation, akin to a filter, to be applied to feature maps. Typically, the size of the pooling operation or filter is smaller than that of the feature map, specifically almost always a 2×2 pixel configuration applied with a 2-pixel stride.

3.2.5 Dropout Layer

In some circumstance the network can behave in weird way by identifying unwanted features and this lead the network to confuse some objects to others. This is what we call overfitting and it is a crucial problem when training a CNN. To alleviate this issue we need to close some neuron in order the network stop to identify unwanted features. This technique is what we call dropout layer. The network through the dropout layer will automatically with respect some probability switch off some neurons.

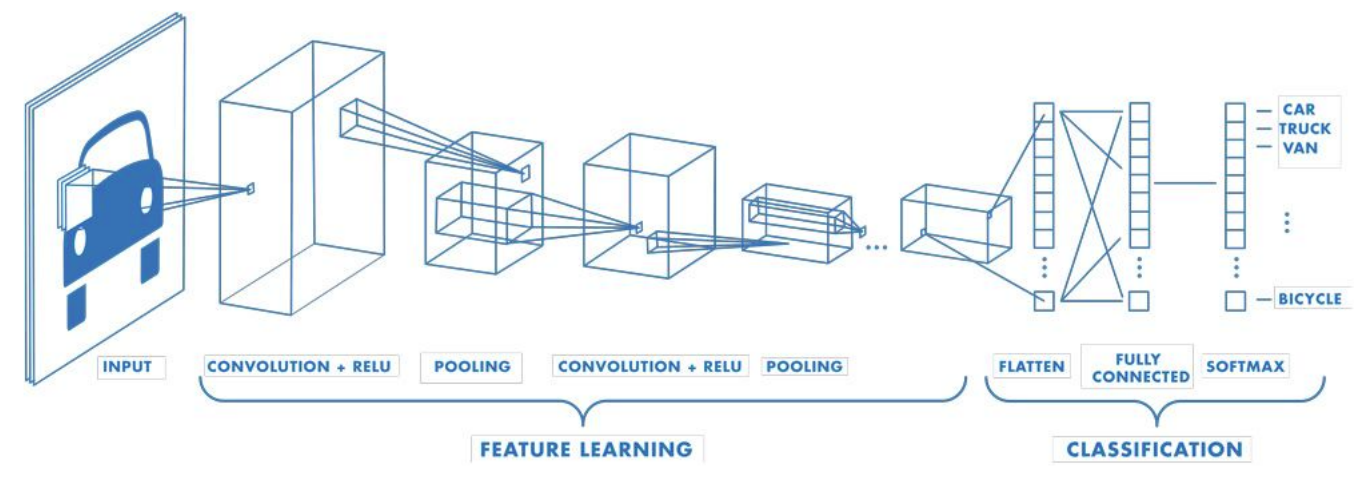
3.2.6 FC Layer

This part discusses the fully connected layer in CNN, and it should not be confused with the fully connected neural network architecture, where all neurons in the input layer are connected

to the neurons in the next layer. This compartment in CNN plays a crucial role in the overall network by predicting the best label to describe the input data. The fully connected layer of a CNN recognizes and classifies the input data and, more importantly, drives the final classification decision. This special compartment of the CNN receives a flattened version of the data coming from either the convolutional or pooling layer. Flattening the data leads to the loss of spatial information, and thus, the fully connected layer operates only on a vectorized representation of the data.

The fully connected layer in CNN contains neurons, and each neuron has its own set of weights. The output of each neuron is computed by applying a linear transformation, which is simply a dot product, followed by a non-linear activation function that allows the network to learn more complex relationships in the data.

It often happens to tune the hyperparameters of the fully connected layer during the network design process. Having more neurons in the layer leads, of course, to more complex representations that the network has to learn in the process. However, the number of parameters drastically increases and may lead to higher computational requirements and possible overfitting if the underlying model is not properly regularized.



Example of a network with many convolutional layers. Filters are applied to each training image at different resolutions, and the output of each convolved image is used as the input to the next layer.

Figure 3.2: The convolutional neural network CNN

4 Proposed Approach

Signal decomposition techniques which in this case are IF and IMFogram have provided two types of datasets. We take advantage of them and build a fusion neural network consisting of a concatenation of ANN and a CNN.

4.1 Artificial neural network (ANN)

Artificial neural network has gained a lot of popularity during recent decades, due to its ability to solve complex problems from financial prediction to machine vision and so on [11]. An ANN is a

combination of different layers (input, hidden and output), activation and loss function.

The structure of an ANN depend on the number of network layers, type of loss and activation function. Choosing different number of layers or different activation functions can generate different models. This work uses a neural network with 4 layers as illustrated in Figure 4.1. Among four layers, we have one input layer, two hidden layers and one output layer [57].

In Figure 4.1 each neuron of the input layer characterize the input feature, neurons in the hidden layers and output layer compute the information obtained from neurons of the previous layer multiplied by the connection weights and apply the activation function to the combined product. The activation function is of great importance, it introduces non-linearity into the network. There exist several activation functions and each has its specificity. We refer the reader to the paper [88] for more details on these functions.

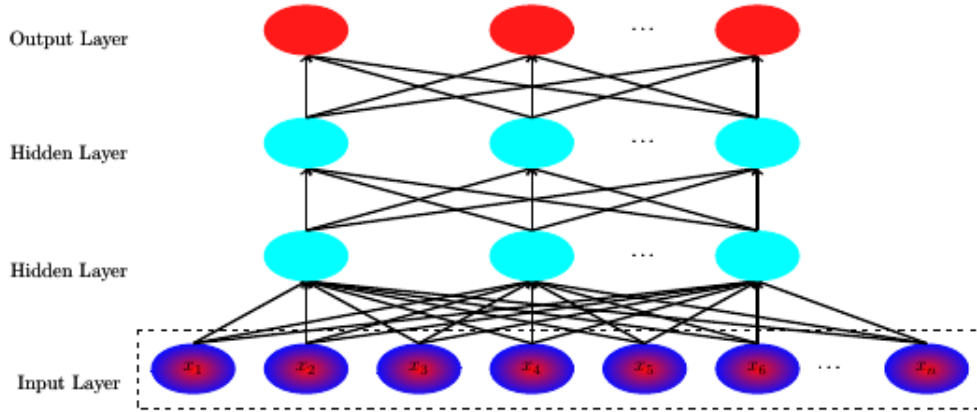


Figure 4.1: Architecture of ANN

To describe how information is processed into the network, we consider that there are n neurons in the input layer, h_1 neurons in the first hidden layer, h_2 in the second hidden layer, and o neurons in the output layer. The forward process of the network is described in the following three steps:

1. As depicted in the input layer of the Figure 4.1, we consider n input features $X = (x_1, \dots, x_n) \in \mathbb{R}^n$. The output of each neurons in the first hidden layer are processed by the following expression:

$$z_j^{H_1} = f(\omega_j^{H_1} \cdot X) = f\left(\sum_{i=1}^n \omega_{ij}^{H_1} x_i\right), \quad j = 1, \dots, h_1 \quad (4.1)$$

where H_1 denote the first hidden layer, and $\omega_j^{H_1} = (\omega_{1j}^{H_1}, \dots, \omega_{nj}^{H_1}) \in \mathbb{R}^n$ are adjustable parameters or weights related to the j th output neuron. $z_j^{H_1}$ is the output value computed at the j th neuron in the first hidden layer and f represent the activation function.

2. The process is similar in the second hidden layer, where in this case the output of the first hidden layer i.e. $z^{H_1} = (z_1^{H_1}, \dots, z_{h_1}^{H_1})$ are the input of the second hidden layer. So we

process neurons in the second hidden layer as follows:

$$z_j^{H_2} = f(\omega_j^{H_2} \cdot z^{H_1}) = f\left(\sum_{i=1}^{h_1} \omega_{ij}^{H_2} z_i^{H_1}\right), \quad j = 1, \dots, h_2 \quad (4.2)$$

where H_2 denote the second hidden layer. We denote by $\omega_j^{H_2} = (\omega_{1j}^{H_2}, \dots, \omega_{h_1j}^{H_2}) \in \mathbb{R}^{h_1}$ is the undetermined weights related to the j th output neuron.

3. Regarding the output layer, the process is similar as of the first and second hidden layer. For this case the output of the second hidden layer, i.e. $z^{H_2} = (z_1^{H_2}, \dots, z_{h_2}^{H_2})$, are the input of the output layer. The neurons of the output layer are computed as follow:

$$z_j^{O_{nn}} = f(\omega_j^{O_{nn}} \cdot z^{H_2}) = f\left(\sum_{i=1}^{h_2} \omega_{ij}^{O_{nn}} z_i^{H_2}\right), \quad j = 1, \dots, o_{nn}, \quad (4.3)$$

where O_{nn} denote the output of the neural networks.

4.2 Convolutional neural network (CNN)

The idea behind CNN is to learn kernels associated to each layer in order to extract feature from the input data. The architecture of the CNN used in this work are depicted in the Figure 4.2 as series of stages. It involves 4 convolutional layers, one flatten layer and two dense layer. The output of each convolutional layer is followed by a pooling, batch normalization, and a dropout layer [59]. Each compartment of the CNN is briefly described as follow:

1. As depicted in the Figure 4.2, the input of the convolutional layer is an image i.e. the IMFogram in the case of this work, which is 3D array. Generally this input is considered to be a 3D array of c channels composed of 2D arrays of size $h \times w$, where each input is characterized by $X \in \mathbb{R}^{c \times h \times w}$. The cross-correlation sometimes referred as a convolutional operation is applied to the c channels and the set of c_0 filter bank also called kernel $K_i^{C_1} \in \mathbb{R}^{c \times t \times t}$ for $1 \leq i \leq c_0$ to generate the local weighted sums. And superscript C_1 refer to the first convolutional layer of CNN. The local weighted sums are encoded in 3D array of \tilde{c} feature maps denoted by $Y \in \mathbb{R}^{\tilde{h} \times \tilde{w}}$, where $\tilde{h} = \frac{h+2p-t}{s} + 1$ and $\tilde{w} = \frac{w+2p-t}{s} + 1$ in the case p stand for a padding parameter and s for a stride parameter. The mathematical expression of the cross-correlation is given as follow:

$$z_j = \sum_{i=1}^{c_0} K_{ij}^{C_1} * X_j \quad \text{where } K_i^{C_1} (1 \leq i \leq c_0) \text{ is a kernel and } X_j (1 \leq j \leq c_1) \text{ the input,} \quad (4.4)$$

more generally for a 2D convolutional operation $*$ it follows

$$z_{j,k,q} = \left(\sum_{i=1}^{c_0} K_{ij}^{C_1} * X_j \right)_{k,q} = \sum_{i=1}^{c_0} \sum_{o_1=1}^t \sum_{o_2=1}^t K_{i,j,o_1,o_2}^{C_1} \cdot X_{j,k+o_1,q+o_2}. \quad (4.5)$$

It is worth noting that the major role of the convolutional layer is to find the local combination of feature from the previous layer [41, 59].

2. The weighted sums are then passed through an activation function f to capture the non-linearity of the model. It worth to acknowledge recent state-of-the-art activation function

such as ReLU [41] and PReLU [89] which solve the vanishing gradient problem [59]. The output of the first convolutional layer after applying an activation function is given as follow:

$$z_j^{C_1} = f(z_j) = f\left(\sum_{i=1}^{c_0} K_{ij}^{C_1} * X_j\right), \quad (4.6)$$

where $i = 1, 2, \dots, c_0$, $j = 1, 2, \dots, c_1$

3. The output of the first convolutional layer are feed to pooling layer as shown in the Figure 4.2. One of the specific role of the pooling layer is to put semantically similar feature into one. This process reduce the dimension of the representation feature and thus creates invariance to small shifts and distortions [41, 59]. The output of the first pooling layer is given by:

$$z^{P_1} = \left(z_1^{P_1}, \dots, z_{p_1}^{P_1}\right), \quad (4.7)$$

where p_1 is the dimension of the output of the first pooling layer.

4. The output of the pooling layer is feed into the batch normalization layer. The batch normalization layer solves the problem related to the change of the distribution of the output layer during training, by performing normalization of each training mini-batch, furthermore accelerates the convergence of learning as well [90, 59]. The output of the first batch normalization layer is given by:

$$z^{B_1} = \left(z_1^{B_1}, \dots, z_{b_1}^{B_1}\right), \quad (4.8)$$

where b_1 is the dimension of the output of the first batch normalization layer.

5. The output of the batch normalization layer is used as an input of the dropout layer as shown in Figure 4.2. It is well known from the literature that the dropout layer is one of the regularization technique to overcome overfitting problem in the CNN. The idea behind a dropout layer process is to randomly drop some neurons and their connections from the network during training [91, 59]. The output of the first dropout layer is given by:

$$z^{D_1} = \left(z_1^{D_1}, \dots, z_{d_1}^{D_1}\right), \quad (4.9)$$

where d_1 is the dimension of the output of the first dropout layer.

6. The output of the dropout layer serve as an input of the second convolutional layer as depicted in the Figure 4.2. The step 1 to 5 are repeated for the rest of compartments until the flatten layer. It worth noting that the output of the second convolutional layer is given as follow:

$$z_j^{C_2} = f\left(\sum_{i=1}^{d_1} K_{ij}^{C_2} * z_j^{D_1}\right), \quad (4.10)$$

where $i = 1, 2, \dots, d_1$, $j = 1, 2, \dots, c_2$, and z^{D_1} the output of the first dropout layer. The output of the third convolutional layer is given as follow:

$$z_j^{C_3} = f \left(\sum_{i=1}^{d_2} K_{ij}^{C_3} * z_j^{D_2} \right), \quad (4.11)$$

where $i = 1, 2, \dots, d_2$, $j = 1, 2, \dots, c_3$, and $z_j^{D_2}$ the output of the second dropout layer.

7. The third batch normalization layer receive input from the third pooling layer and normalize this input. The out put of the third batch normalization layer is given by:

$$z^{B_3} = (z_1^{B_3}, \dots, z_{b_3}^{B_3}) \quad (4.12)$$

where b_3 is the output dimension of the batch normalization layer. This output serve as input of the third dropout layer

8. The output of third dropout layer as depicted in the Figure 4.2 is used as an input of the flatten layer. Flatten layer transform data into a 1D array, which is of a tremendous importance in this study since we are predicting a time series which is also a 1D array. After applying the dropout process the output is given by :

$$z^{D_3} = (z_1^{D_3}, \dots, z_{d_3}^{D_3}), \quad (4.13)$$

where d_3 is the number of neurons remained after the dropout process.

9. The output of the flatten layer is a 1D array and serves as input to the dense layer depicted in the Figure 4.2. After flattened the output of the third dropout layer, the output of the flatten layer is given by:

$$z^{F_l} = (z_1^{F_l}, \dots, z_{f_l}^{F_l}), \quad (4.14)$$

where f_l is the dimension of the obtained array after the flatten process.

10. To compute the output of the dense layer, we first precise that the input of the dense layer is the output of te flatten layer i.e. $z^{F_l} = (z_1^{F_l}, \dots, z_{f_l}^{F_l})$, therefore the output of the dense layer is given by:

$$z_j^{D_l} = f(\omega_j^{D_l} \cdot z^{F_l}) = f \left(\sum_{i=1}^{f_l} \omega_{ij}^{D_l} \cdot z_i^{F_l} \right), \quad (4.15)$$

where $j = 1, 2, \dots, d_l$, f is the activation function and $\omega_j^{D_l}$ the weights associated to the dense layer depicted in the Figure 4.2.

11. The output of the fourth dropout layer is given by

$$z^{D_4} = (z_1^{D_4}, \dots, z_{d_4}^{D_4}), \quad (4.16)$$

where the input is the output of the dense layer as depicted in the Figure 4.2 i.e. $z_j^{D_l}$ for $j = 1, 2, \dots, d_l$, and d_4 is the output dimension after the dropout process.

12. Regarding the output layer, the process is similar as of the output of the ANN. For this case the output of the fourth dropout layer, i.e. $z^{D_4} = (z_1^{D_4}, \dots, z_{d_4}^{D_4})$, are the input of the output layer and d_4 is the number of neuron in the fourth dropout layer. The neurons of the output layer are computed as follow:

$$z_j^{O_{cn}} = f(\omega_j^{O_{cn}} \cdot z^{D_4}) = f \left(\sum_{i=1}^{d_4} \omega_{ij}^{O_{cn}} z_i^{D_4} \right), \quad j = 1, \dots, o_{cn} \quad (4.17)$$



Figure 4.2: The Architecture of the CNN

4.3 Fusion neural network (FNN)

The theory of statistical learning stipulates that, for two given datasets X and Y in topological spaces \mathcal{X} and \mathcal{Y} , respectively, predicting Y based on X is meaningful when Y depends non-trivially on X . This task becomes easier when we have knowledge of the conditional distribution of Y given X , or when we employ statistical techniques to estimate Y given X (Rigollet, [92]). This work leverages two types of datasets generated using signal processing techniques which depend non-trivially on the FTS and utilizes innovative machine learning techniques to predict FTS.

The novelty of this study lies in the fusion of two distinct neural networks to predict FTS. The fusion neural network build in this work, takes two different sets of input data, each fed into separate neural networks. One component of the Fusion Neural Network (FNN) is the Artificial Neural Network (ANN), responsible for processing the 1D data generated by the Intrinsic Mode Function (IMF) decomposition of the signal produced using the IF algorithm. The other component of the FNN is the Convolutional Neural Network (CNN), which deals with the images of the time-frequency representation of FTS. In this work we choose to use the IMFogram. The component of the Figure 4.3 is described in the following steps:

1. The output of the ANN as depicted in Figure 4.1 and in point 3 of Section 4.1 is concatenated with the output of the CNN depicted in Figure 4.2 and in point 8 of Section 4.2 to serve as input to the architecture illustrated in Figure 4.3 which constitute the last block of the FNN.

The concatenation makes sense since the output of the ANN is a 1D array and the output of the CNN is also a 1D array as explained in point 8 of Section 4.2.

2. The 1D concatenated array serves as input of the dense layer. Generally the output of the dense layer is computed as a dot product between the concatenated 1D array and the weight matrix $W_i^{D_F}$, to which a bias vector b_i is added. We denote the concatenated 1D array by $X_i^F = (z_1^{O_{nn}}, \dots, z_{o_{nn}}^{O_{nn}}, z_1^{O_{cn}}, \dots, z_{o_{cn}}^{O_{cn}})$. It worth to precise again that $(z_1^{O_{nn}}, \dots, z_{o_{nn}}^{O_{nn}})$ is the output of the ANN and $(z_1^{O_{cn}}, \dots, z_{o_{cn}}^{O_{cn}})$ is the output of the CNN, therefore, the output of dense layer is given by:

$$y_i^{D_F} = f(W_i^{D_F} \cdot X_i^F) = f\left(\sum_{j=1}^{d_c} W_{ij}^{D_F} \cdot X_j^F\right) \quad (4.18)$$

where f represent the activation function and $d_c = o_{nn} + o_{cn}$ the dimension of the concatenated 1D array, where o_{nn} is the dimension of the output of the ANN and o_{cn} is the dimension of the output of the CNN.

3. The output of the dense layer is used as the input of the output layer. The process is similar as of the dense layer, but in this case the input of the output layer of the FNN is the vector $(y_1^{D_F}, \dots, y_{o_F}^{D_F})$. The neurons of the output layer of FNN are computed as follow:

$$y_i^{O_F} = f(W_i^{O_F} \cdot y_i^{D_F}) = f\left(\sum_{j=1}^{d_F} W_{ij}^{O_F} \cdot y_j^{D_F}\right), \quad 1 \leq i \leq o_F, \quad (4.19)$$

where $W_i^{O_F} = (W_{i1}^{O_F}, \dots, W_{id_F}^{O_F})$ is the undetermined weights related to the i th output neuron, f the activation function and d_F is the output dimension of the previous layer.

4. The neurons of the output layer are fed into the loss layer. In general different loss functions can be used in the loss layer depending on the required task to compute the error between the estimated and the target value. Among the list of loss functions there is the squared loss, softmax, binary cross-entropy [57]. The main loss function used for regression task is the squared loss

$$\Theta(y_i^{O_F}, y_i) = \frac{1}{2}(y_i^{O_F} - y_i)^2. \quad (4.20)$$

Binary cross-entropy loss is the main loss function for the classification task. It is defined by:

$$\Theta(y_i^{O_F}, y_i) = -y_i \log(y_i^{O_F}) - (1 - y_i) \log(1 - y_i^{O_F}). \quad (4.21)$$

The loss function in the loss layer is therefore computed as follows:

$$L(x, y) = \sum_{i=1}^{o_F} \Theta(y_i^{O_F}, y_i) + \frac{\alpha}{2} \sum_{j=1}^P \|\theta_j\|^2 \quad (4.22)$$

where:

- $\Theta(y_i^{O_F}, y_i)$ is the primary loss, which measures the difference between the predicted and the target value.
- $\frac{\lambda}{2}$ is the regularization parameter, which control the effect of regularization. This value is choosed in this work according to the hyper-parameter optimization.
- $\sum_{j=1}^P \|\theta_j\|^2$ is the sum of the square of all the parameters in the fusion neural network (i.e. parameter of ANN, CNN and including the one in Figure 4.3). P represents the total number of parameters in the FNN.

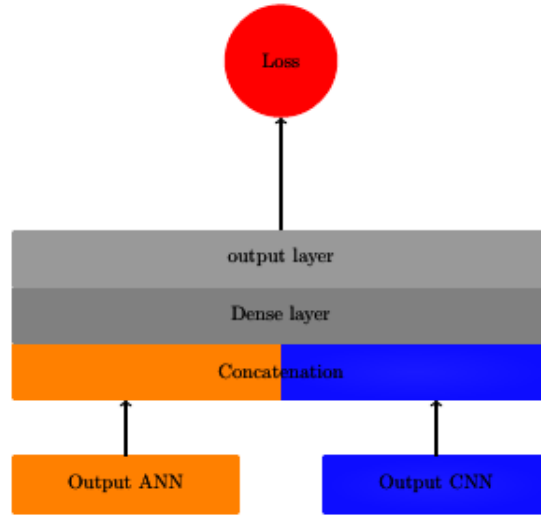


Figure 4.3: The output of the FNN

4.4 Back propagation of FNN

The essence of learning in neural network is the back propagation. It is a fundamental algorithm that train the network. It is also referred as a supervised algorithm used to update parameters during training and thus provide the network with the ability to make more accurate predictions. All parameters in the FNN are updated with respect to stochastic gradient descent (SGD) method within back propagation process [57]. The learning process through back propagation is described in the following steps:

1. The computation of gradient of the input and weights of output layer of FNN is given by:

$$\frac{\partial L}{\partial y_i^{O_F}} = \frac{\partial \mathcal{L}}{\partial y_i^{O_F}}, \quad \frac{\partial L}{\partial W_{ij}^{O_F}} = \frac{\partial L}{\partial y_i^{O_F}} \cdot \frac{\partial f}{\partial \beta} \cdot y_i^{D_F} + \alpha \cdot W_{ij}^{O_F} \quad (4.23)$$

where $\beta = W_i^{O_F} \cdot y_i^{D_F}$. The weights of the output layer of FNN are updates as follows:

$$W_{ij}^{O_F} = W_{ij}^{O_F} - \eta \frac{\partial L}{\partial W_{ij}^{O_F}} \quad (4.24)$$

where η represent the learning rate, and $1 \leq i \leq d_F$, $1 \leq j \leq o_F$.

2. To update the weights of the dense layer depicted in figure 4.3, it is necessary to first compute the gradient of the input and weights of the dense layer in the figure 4.3 as follows:

$$\frac{\partial L}{\partial y_j^{D_F}} = \sum_{i=1}^{o_F} \frac{\partial L}{\partial y_i^{O_F}} \cdot \frac{\partial f}{\partial \beta} \cdot W_{ij}^{O_F} \quad (4.25)$$

where $\beta = W_j^{O_F} \cdot y_i^{D_F}$ and X_i^F is the 1D concatenated array formed by the output of the ANN depicted in the figure 4.1 and CNN depicted in the figure 4.2. The gradient with respect to

weights of the dense layer in the figure 4.3 is given by:

$$\frac{\partial L}{\partial W_{ij}^{D_F}} = \frac{\partial L}{\partial y_j^{D_F}} \cdot \frac{\partial f}{\partial \beta} X_i^F + \alpha \cdot W_{ij}^{D_F}, \quad (4.26)$$

where $\beta = W_i^{D_F} \cdot X_i^F$ the dot product between the weights of the dense layer in the figure 4.3 and the concatenated 1D array. The weights of the dense layer in figure 4.3 are thus updated as follow:

$$W_{ij}^{D_F} = W_{ij}^{D_F} - \eta \cdot \frac{\partial L}{\partial W_{ij}^{D_F}} \quad (4.27)$$

where $i = 1, 2, \dots, d_c$, $j = 1, 2, \dots, d_F$, and η is the learning rate.

3. This step update simultaneously the weights of the output layer of the ANN and the CNN depicted in the figure 4.1 and 4.2 respectively. For the ANN we have:

$$\frac{\partial L}{\partial z_j^{O_{nn}}} = \sum_{i=1}^{o_{nn}} \frac{\partial L}{\partial y_i^{D_F}} \cdot \frac{\partial f}{\partial \beta} \cdot W_{ij}^{D_F} \quad (4.28)$$

where $\beta = \sum_{i=1}^{o_{nn}} W_{ij}^{D_F} X_i^F = \sum_{i=1}^{o_{nn}} W_{ij}^{D_F} z_i^{O_{nn}}$, this is the case due to equation (4.18), where $X_i^F = (z_1^{O_{nn}}, \dots, z_{o_{nn}}^{O_{nn}}, z_1^{O_{cn}}, \dots, z_{o_{cn}}^{O_{cn}})$ and the contribution of the ANN to the dense layer depicted in figure 4.3 is $\sum_{i=1}^{o_{nn}} W_{ij}^{D_F} z_i^{O_{nn}}$. For the CNN we have:

$$\frac{\partial L}{\partial z_j^{O_{cn}}} = \sum_{i=1}^{o_{cn}} \frac{\partial L}{\partial y_i^{D_F}} \cdot \frac{\partial f}{\partial \beta} \cdot W_{ij}^{D_F}, \quad (4.29)$$

where $\beta = \sum_{i=1}^{o_{cn}} W_{ij}^{D_F} z_i^{O_{cn}}$ which represent the contribution of the CNN to the dense layer shown in the figure 4.3. The gradient of the loss function with respect to the weights of the output layer of the ANN is computed as follows:

$$\frac{\partial L}{\partial \omega_{ij}^{O_{nn}}} = \frac{\partial L}{\partial z_j^{O_{nn}}} \cdot \frac{\partial f}{\partial \beta} \cdot z_j^{H_2} + \alpha \omega_{ij}^{O_{nn}} \quad (4.30)$$

where $\beta = \omega_j^{O_{nn}} \cdot z_j^{H_2}$ the dot product between the weights of the output layer and the input of the output layer of the ANN. The weights of the output layer of the ANN are updated as follows:

$$\omega_{ij}^{O_{nn}} = \omega_{ij}^{O_{nn}} - \eta \frac{\partial L}{\partial \omega_{ij}^{O_{nn}}} \quad (4.31)$$

where $i = 1, 2, \dots, h_2$, $j = 1, 2, \dots, o_{nn}$, and η is the learning rate. For the output of the CNN, the gradient of the loss function with respect to the weights of the output layer of the CNN is computed as follows:

$$\frac{\partial L}{\partial \omega_{ij}^{O_{cn}}} = \frac{\partial L}{\partial z_j^{O_{cn}}} \cdot \frac{\partial f}{\partial \beta} \cdot z_j^{D_4} + \alpha \omega_{ij}^{O_{cn}} \quad (4.32)$$

where $\beta = \omega_j^{O_{cn}} \cdot z^{D_4}$ the dot product between the weights of the output layer and the input of the output layer of the CNN. The weights of the output layer of the CNN are updated as follows:

$$\omega_{ij}^{O_{cn}} = \omega_{ij}^{O_{cn}} - \eta \frac{\partial L}{\partial \omega_{ij}^{O_{cn}}} \quad (4.33)$$

where $i = 1, 2, \dots, d_4$, $j = 1, 2, \dots, o_{cn}$, and η is the learning rate.

4. This step update simultaneously the second hidden layer of the ANN as depicted in figure 4.1 and the dense layer of the CNN according to the figure 4.2. For the second layer of the ANN the gradient with respect to the output of the second layer is given as follows:

$$\frac{\partial L}{\partial z_j^{H_2}} = \sum_{i=1}^{o_{nn}} \frac{\partial L}{\partial z_i^{O_{nn}}} \cdot \frac{\partial f}{\partial \beta} \cdot \omega_{ij}^{O_{nn}}, \quad (4.34)$$

where $\beta = \omega_j^{O_{nn}} \cdot z^{H_2}$ the dot product between the weights of the output layer and the input of the output layer of the ANN. For the dense layer of the CNN, we have:

$$\frac{\partial L}{\partial z_j^{D_l}} = \sum_{i=1}^{d_4} \frac{\partial L}{\partial z_i^{D_4}} \frac{\partial z_i^{D_4}}{\partial z_j^{D_l}}. \quad (4.35)$$

The gradient of the loss function with respect to the weights of the second hidden layer of the ANN is given by:

$$\frac{\partial L}{\partial \omega_{ij}^{H_2}} = \frac{\partial L}{\partial z_j^{H_2}} \cdot \frac{\partial f}{\partial \beta} \cdot z_j^{H_1} + \alpha \omega_{ij}^{H_2}, \quad (4.36)$$

where $\beta = \omega_{ij}^{H_2} \cdot z_j^{H_1}$ the dot product between the weights of the second hidden and the input layer of the ANN. The weights of the second hidden layer are therefore updated as follows:

$$\omega_{ij}^{H_2} = \omega_{ij}^{H_2} - \eta \frac{\partial L}{\partial \omega_{ij}^{H_2}} \quad (4.37)$$

where $i = 1, 2, \dots, h_1$, $j = 1, 2, \dots, h_2$, and η is the learning rate. For the dense layer of the CNN, the gradient of the loss function with respect to the weights of the dense layer is computed as follows:

$$\frac{\partial L}{\partial \omega_{ij}^{D_l}} = \frac{\partial L}{\partial z_j^{D_l}} \cdot \frac{\partial f}{\partial \beta} \cdot z_j^{F_l} + \alpha \omega_{ij}^{D_l} \quad (4.38)$$

where $\beta = \omega_{ij}^{D_l} \cdot z_j^{F_l}$ the dot product between the input and the weights of the dense layer of the CNN as shown in the figure 4.2. We thus update the weights of the dense layer as follow:

$$\omega_{ij}^{D_l} = \omega_{ij}^{D_l} - \eta \frac{\partial L}{\partial \omega_{ij}^{D_l}} \quad (4.39)$$

where $i = 1, 2, \dots, f_l$, $j = 1, 2, \dots, d_l$, and η is the learning rate.

5. This step update simultaneously the first hidden layer of the ANN as depicted in figure 4.1 and the third convolutional layer of the CNN according to the figure 4.2. For the first layer of the ANN the gradient with respect to the output of the second layer is given as follows:

$$\frac{\partial L}{\partial z_j^{H_1}} = \sum_{i=1}^{h_2} \frac{\partial L}{\partial z_i^{H_2}} \cdot \frac{\partial f}{\partial \beta} \cdot \omega_{ij}^{H_2}, \quad (4.40)$$

where $\beta = \omega_j^{H_2} \cdot z^{H_1}$ the dot product between the weights of the output layer and the input of the output layer of the ANN. For the third convolutional layer of the CNN, we have:

$$\frac{\partial L}{\partial z_j^{C_1}} = \sum_{i=1}^{d_4} \frac{\partial L}{\partial z_i^{P_3}} \cdot \frac{\partial z_i^{P_3}}{\partial z_j^{C_1}}. \quad (4.41)$$

The gradient of the loss function with respect to the weights $\omega_{ij}^{H_1}$ is given by:

$$\frac{\partial L}{\partial \omega_{ij}^{H_1}} = \frac{\partial L}{\partial z_j^{H_1}} \cdot \frac{\partial f}{\partial \beta} \cdot \omega_j^{H_1} + \alpha \omega_{ij}^{H_1} \quad (4.42)$$

where $\beta = \omega_{ij}^{H_1} \cdot X$ the dot product between the weights of the first hidden and the input data of the ANN. The weights of the first hidden layer are therefore updated as follow:

$$\omega_{ij}^{H_1} = \omega_{ij}^{H_1} - \eta \frac{\partial L}{\partial \omega_{ij}^{H_1}} \quad (4.43)$$

where $i = 1, 2, \dots, n$, $j = 1, 2, \dots, h_1$, and η is the learning rate. For the third convolutional layer of the CNN, the gradient of the loss function with respect to the kernel of the third convolutional layer is computed as follow:

$$\frac{\partial L}{\partial K_{ij}^{C_3}} = \frac{\partial L}{\partial z_j^{C_3}} \cdot \frac{\partial f}{\partial \beta} \cdot \frac{\partial \left(\sum_{i=1}^{d_2} K_{ij}^{C_3} * z_i^{D_2} \right)}{\partial K_{ij}^{C_3}} + \alpha K_{ij}^{C_3} \quad (4.44)$$

where $\beta = \sum_{i=1}^{c_2} K_{ij}^{C_3} * z_i^{D_2}$ the convolution between the kernel of the third convolutional layer and the output of the second dropout layer as depicted in the figure 4.2. The kernels of the first hidden layer are therefore updated as follow:

$$K_{ij}^{C_3} = K_{ij}^{C_3} - \eta \frac{\partial L}{\partial K_{ij}^{C_3}} \quad (4.45)$$

where $i = 1, 2, \dots, d_2$, $j = 1, 2, \dots, c_3$, and η is the learning rate.

6. The update of the first and second convolutional layer of the CNN are done in a similar way as in step 5. The update for the second convolutional layer is given by:

$$K_{ij}^{C_2} = K_{ij}^{C_2} - \eta \frac{\partial L}{\partial K_{ij}^{C_2}} \quad (4.46)$$

where $i = 1, 2, \dots, d_1$, $j = 1, 2, \dots, c_2$, and η is the learning rate. The update for the first convolutional layer is given by:

$$K_{ij}^{C_1} = K_{ij}^{C_1} - \eta \frac{\partial L}{\partial K_{ij}^{C_1}} \quad (4.47)$$

where $i = 1, 2, \dots, c_0$, $j = 1, 2, \dots, c_1$, and η is the learning rate.

5 Datasets and the simulation results

5.1 Datasets

This work leverages two types of datasets generated by IF and IMFogram. As illustrated in figures 5.4 and 5.5, one can quickly observe some unusual effects in both figures. These effects stem from the boundary effects. Naturally, these effects arise because FTS is a non-stationary signal generated by a nonlinear source, and IF is used for its decomposition, which involves the subsequent subtraction of the original signal to obtain the IMFs [93]. The actual problem arises from IF which requires periodicity at the boundary of the signal.

To address this issue, careful consideration of the boundary conditions is required, as both datasets are based on a IF decomposition. To address boundary conditions, assumptions must be made regarding the extension of the signal to the right and left. In other words, it is necessary to extrapolate the FTS beyond its boundaries. Given that the signal has undergone decomposition using IF-based methods, there exists the opportunity to select an optimal extension to minimize errors associated with end effects in the decomposition process [93].

The signal can be extended periodically using symmetric (reflexive), anti-symmetric (anti-reflexive) boundary extension, or a combination of both. We proceed to analyze each extension in conjunction with the corresponding IMFs and IMFogram in the subsequent steps.

1. **Anti-Symmetric (Anti-Reflective) extension:** Consider a signal S of length n , and let L represent the extension length beyond the signal S . To extend the signal S in an anti-symmetric manner beyond its boundary. The end point of the signal is used as point of symmetry like the origin for an odd function.

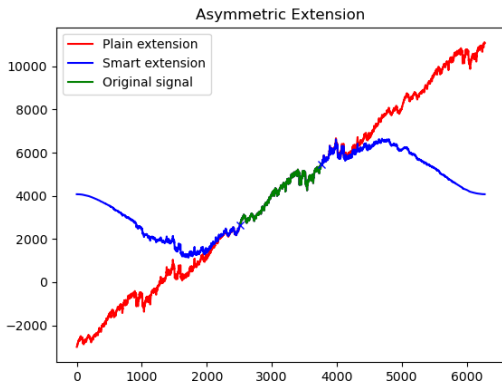


Figure 5.1: Anti-symmetric extension of Nasdaq

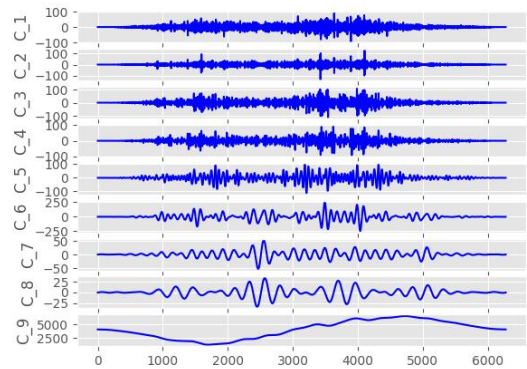


Figure 5.2: IMFs of the Anti-symmetric extension

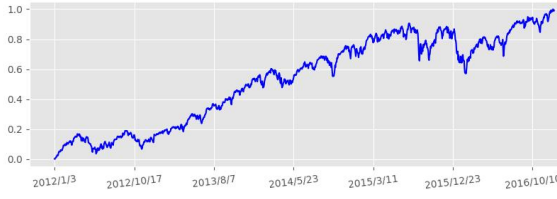


Figure 5.3: NASDAQ time series from January 4th 2012 to December 30th 2016

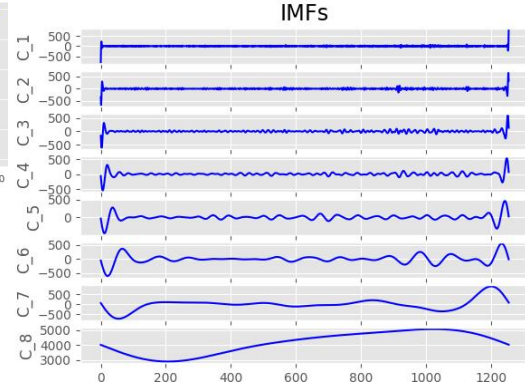


Figure 5.4: The IMFs of Nasdaq Financial time series

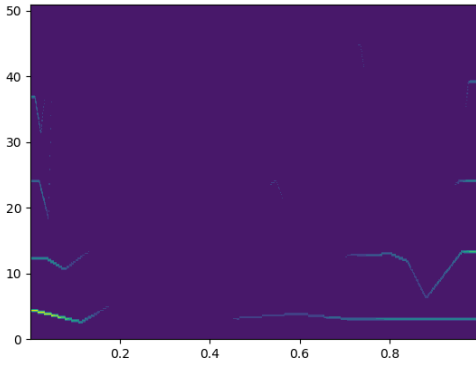


Figure 5.5: The IMFogram of the figure 5.3 and 5.4

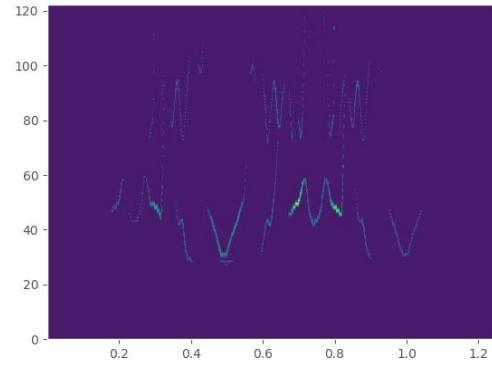


Figure 5.6: IMFogram generated from the IMFs of the Anti-symmetric extension

It is worth noting, as emphasized in [93], that despite performing the extension, some end effects may persist. To tackle this issue, one approach is to enforce the extended signal to become periodic at the newly generated boundaries. Let's refer to this as the "smart extension", denoting the forced periodic signal obtained from the extended signal. The following steps outline the key procedures to obtain the smart extended signal:

- Calculate the mean value m of the original signal S and subsequently subtract it from the signal.
- The subtracted signal, $S - m$, is extended beyond the boundaries, resulting in an extended signal denoted as S_{ext} .
- After obtaining the signal S_{ext} , we then multiply it by a characteristic function that takes on a value of one in the interval corresponding to the original signal S and smoothly diminishes to zero as we approach the new boundaries of S_{ext} .
- The smart extended signal is obtained by reintroducing the mean value m of the original signal [93].

$$S_{\text{new}} = \mathcal{X} \cdot S_{\text{ext}} + m. \quad (5.1)$$

Figure 5.1 illustrates the anti-symmetric extension of the Nasdaq data. As observed, the original signal is represented in green, the anti-symmetric signal in red, and the smart

extension, built upon the anti-symmetric extension, is depicted in blue. The smart extension is two times longer than the original signal, $L = 2 \times \text{length}(S)$.

Figure 5.2 illustrates the Intrinsic Mode Functions (IMFs) of the smart extension signal. Notably, there are discernible differences between the IMFs of the original signal depicted in Figure 5.4 and those of the smart extension shown in Figure 5.2. The primary distinction lies in the number of IMFs generated, with 9 IMFs in Figure 5.2 compared to 8 IMFs in Figure 5.4. Additionally, the end effects visible in Figure 5.4, especially in the first three IMFs, are absent in Figure 5.2.

Furthermore, the IMFogram figures corresponding to the two sets of IMFs also exhibit differences, as shown in Figure 5.5 and Figure 5.6. The end effects seen in Figure 5.4 have repercussions on the IMFogram in Figure 5.5, noticeable in the higher energies on the two horizontal lines. This contrast is not observed in Figure 5.6.

2. **Symmetric (Reflective) extension or the Neumann boundary condition:** Consider the signal S with indices ranging from 1 to n , representing a signal of length n . Let L denote the length by which we intend to extend S beyond its boundaries. To symmetrically extend it, we assume that the data outside S mirror a reflection of the data inside S [94]. For instance, on the left-hand side of the signal S , the first element in the extension, denoted as S_0 , is equal to the left boundary point of S (S_1). Similarly, the second element of the left extension, denoted as S_{-1} , is equal to the second element of S (S_2), and so forth. The same principle applies to the right symmetric extension of S . A precise mathematical description is provided below:

$$S_{1-j} = S_j \quad \text{for } j = 1, 2, \dots, L, \quad (5.2)$$

the left extension of S , and the right extension is given as

$$S_{n+j} = S_{n+1-j} \quad \text{for } j = 1, 2, \dots, L, \quad (5.3)$$

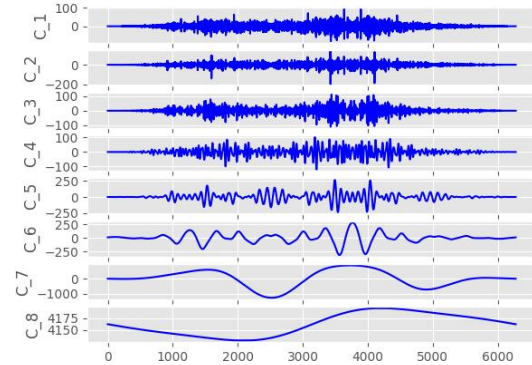
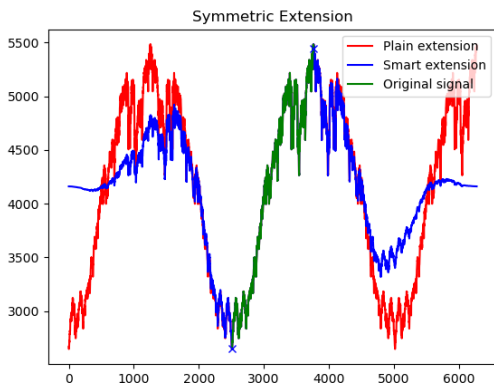


Figure 5.7: symmetric extension of Nasdaq **Figure 5.8:** IMFs of the symmetric extension

The formulas outlined in (5.2) and (5.3) illustrate the process of symmetrically extending a given signal. Following the application of these formulas, the smart extension, as described in points (a) to (d) above, can be applied. This step enforces the symmetric extension to become periodic at the boundaries, mitigating any remaining end effects.

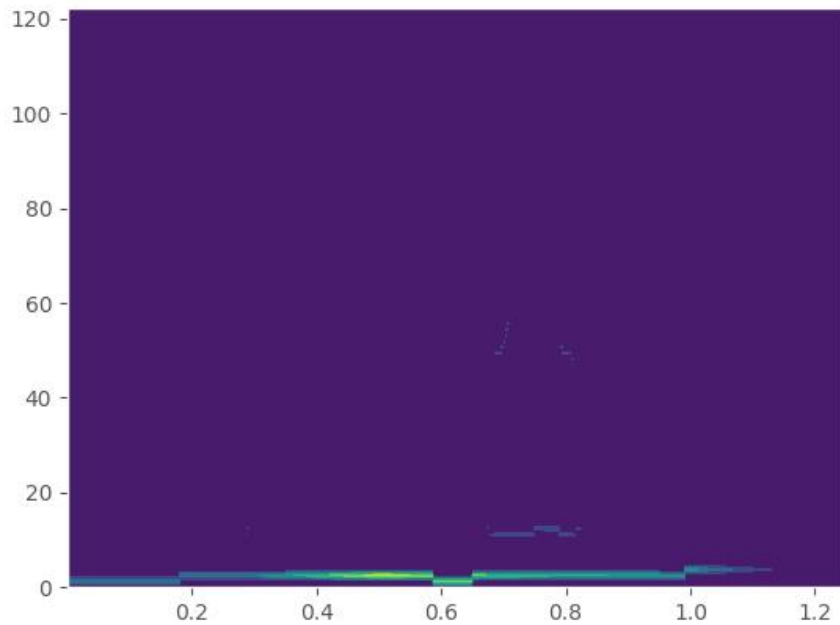


Figure 5.9: IMFogram generated from the IMFs of the symmetric extension

Figure 5.7 illustrates the Nasdaq symmetric extension alongside the smart symmetric extension. The Nasdaq signal is presented in green, its symmetric extension in red, and the smart symmetric extension in blue. Figure 5.8 depicts the IMFs generated from the smart symmetric extension of the Nasdaq time series. Once again, a noticeable difference can be observed between the IMFs generated from the original Nasdaq time series (Figure 5.4) and those generated from the smart symmetric extension (Figure 5.8). Notably, there are no end effects in Figure 5.8, which contrasts with the presence of end effects in Figure 5.4.

Furthermore, Figure 5.9 illustrates the IMFogram derived from the IMFs generated by the smart symmetric extension, where higher energies manifest in the low frequencies.

3. **Anti-symmetric and Symmetric extension:** The anti-symmetric and symmetric extension is a hybrid approach that involves a combination of both types of extension. The algorithm initiates by extending the signal anti-symmetrically on the left, with the length of the extension equal to the length of the Nasdaq signal (i.e., $\text{length}(S) = 1255$). Subsequently, after the 1255th left anti-symmetric extension, the algorithm proceeds to extend the signal symmetrically on the left once again, also with a length of 1255.

This can be clearly observed in Figure 5.10, where the red signal represents the anti-symmetric and symmetric extension of the Nasdaq time series. Specifically, from point 1255 to 2510 on the horizontal axis, the signal undergoes anti-symmetric extension (mirroring the appearance in Figure 5.1 from point 1255 to 2510). Subsequently, the signal experiences symmetric extension from point 0 to point 1255 on the horizontal axis, highlighting the symmetry of the signal segment from point 1255 to 2510 on the horizontal axis.

The right extension in Figure 5.10 follows a similar procedure. In this instance, the signal is initially extended symmetrically from points 3765 to 5020 on the horizontal axis and subsequently extended anti-symmetrically from points 5020 to 6275. The Nasdaq signal

is represented by the green curve in Figure 5.10, while the blue curve corresponds to the smart extension derived from the anti-symmetric and symmetric extension process.

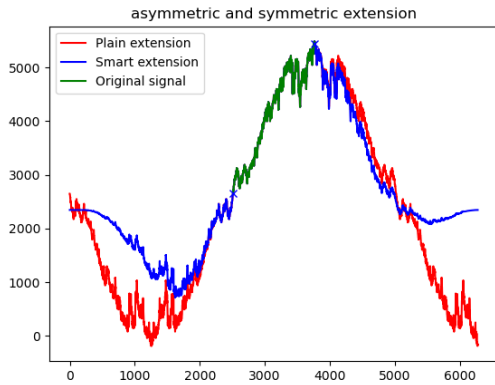


Figure 5.10: Asymmetric and Symmetric extension of Nasdaq

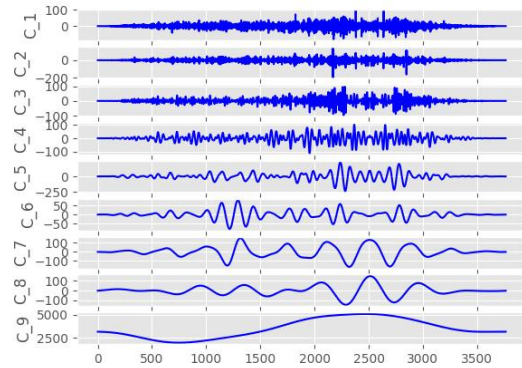


Figure 5.11: IMFs of the anti-symmetric symmetric extension

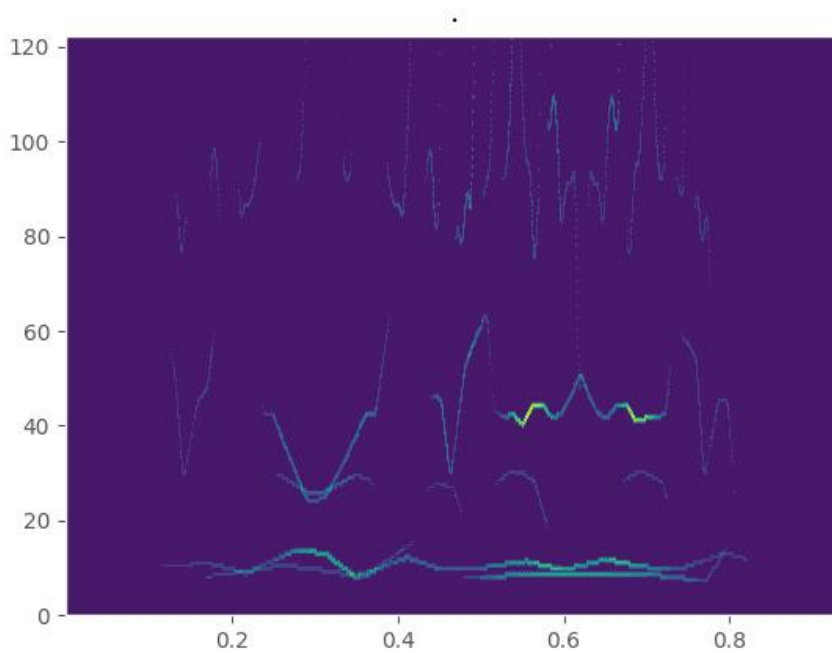


Figure 5.12: IMFogram generated from the IMFs of the anti-symmetric and symmetric extension

Figure 5.11 illustrates the IMFs generated from the smart anti-symmetric and symmetric extension. Once again, a noticeable difference can be observed between the IMFs generated from the original Nasdaq time series, as depicted in Figure 5.4, and those generated by the smart anti-symmetric and symmetric extension, shown in Figure 5.11. Figure 5.11 displays 9 IMFs without any detectable boundary effects.

Figure 5.12 depicts the IMFogram of the IMFs generated by the smart anti-symmetric and

symmetric extension of the Nasdaq time series. This figure illustrates the distribution of higher energies from low to high frequencies, with no discernible end effects.

4. **Symmetric and Anti-symmetric extension:** The symmetric and anti-symmetric extension operates similarly to the anti-symmetric and symmetric extension explained in point 3 above. In this scenario, the signal on the left side of the Nasdaq time series, depicted in green in Figure 5.13, is initially extended symmetrically from point 1255 to 2510 on the horizontal axis. This symmetric extension is represented by the red signal in Figure 5.13, effectively mirroring the green portion of the signal. Subsequently, the remaining red portion of the signal from point 0 to 1255 undergoes an anti-symmetric extension, completing the transformation.

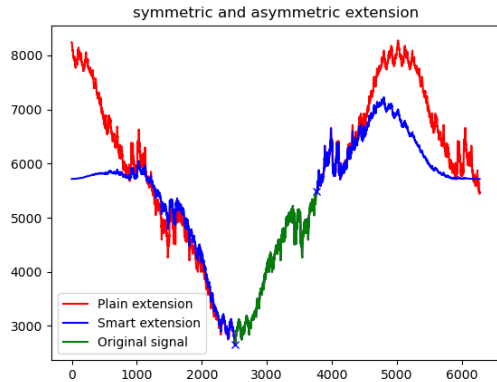


Figure 5.13: Asymmetric and Symmetric extension of Nasdaq

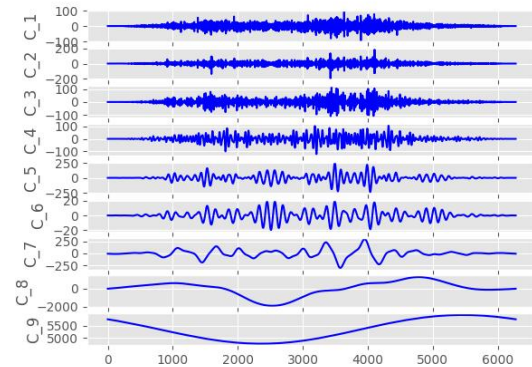


Figure 5.14: IMFs of the symmetric and anti-symmetric extension

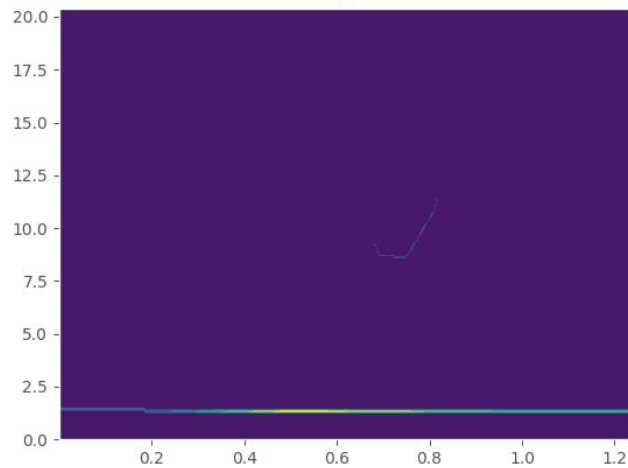


Figure 5.15: IMFogram generated from the IMFs of the symmetric and anti-symmetric extension

The right symmetric and anti-symmetric extension of the Nasdaq time series, depicted in green in Figure 5.13, follows a similar procedure as the left symmetric and anti-symmetric extension. In this case, the signal is initially extended anti-symmetrically from points 3765 to 5020 and then symmetrically from points 5020 to 6275.

Figure 5.14 illustrates the IMFs generated from the symmetric and anti-symmetric extension of the Nasdaq time series. This figure displays 9 IMFs without any apparent end effects, whereas Figure 5.4 has 8 IMFs with visible end effects.

Figure 5.15 illustrates the IMFogram associated with the smart symmetric and anti-symmetric extension. This IMFogram depicts high energies in the low frequencies. A notable distinction between the smart symmetric and anti-symmetric extension and the smart anti-symmetric and symmetric extension can be readily observed by comparing the IMFograms presented in Figure 5.15 and Figure 5.12.

5.2 Simulation Results

The data utilized for the experiments comprises the National Association of Securities Dealers Automated Quotations (NASDAQ) index. The total number of NASDAQ values is 1214, collected from January 4th, 2012, to December 30th, 2016. Figure 5.3 displays the original time series. The data is divided into training sets (70% of the total trading days), evaluation sets (20% of the total trading days), and testing sets (10% of the total trading days). The data's statistics are detailed in Table 5.1, where N_{all} and N_{out} represent the size of the entire dataset and the combination of the testing and evaluation samples, respectively.

Table 5.1: The data and the descriptive statistical analysis

Name	N_{all}	N_{out}	Mean	Std.	Data range
NASDAQ	1214	364	4161.88	825.13	2012.01.04-2021.12.30

5.2.1 Experimental setting

The proposed model is a hybrid system that initially employs the IF algorithm and a fusion neural network for predicting FTS. The IF algorithm is implemented using Matlab¹, while the fusion neural network is developed using the Keras² platform. The execution of the fusion neural network takes place on Google Colab utilizing GPUs.

For training the suggested fusion neural network model, we employed Nesterov-based stochastic gradient descent (SGD) [46] to adjust the weights. Each weight is updated with the inclusion of a momentum parameter, which regulates the influence of the historical gradient direction on the current gradient descent direction. To mitigate overfitting and enhance the model's stability, we implemented L_2 regularization. Additionally, we incorporated a weights decay parameter to account for the role of the regularization term in the neural networks.

The IF algorithm decomposes the time series into M Intrinsic Mode Functions (IMFs). As suggested in [57], it is practically advantageous to consider a set of L consecutive values from each IMF as the input samples. For the input of the Artificial Neural Network (ANN) segment in the proposed fusion neural networks, we set L to 4. Consequently, the input sample size of the ANN is (1213, 4).

¹IF code:<https://github.com/Acicone/FIF>, Jan. 2020.

²Website of the Keras platform:<https://keras.io>, Jan. 2020.

To cater to the input requirements of the Convolutional Neural Network (CNN) segment, we reshape the IMFogram into a tensor of size (1213, 25, 25, 1). This reshaping is done to align with the size of the label, which is a one-dimensional vector of size (1213, 1) extracted from the original NASDAQ signal. It is essential to clarify that if the first dimension of the input sample for the ANN differs from that of the input sample for the CNN, and the label, the fusion neural network will not function correctly.

Regarding the hyperparameters described in Section 4 and those employed in the Nesterov-based Stochastic Gradient Descent (SGD) method, we allowed Hyperopt³ to autonomously determine their values on the validation datasets. Hyperopt, a Python library, serves as a tool for both serial and parallel optimization within complex search spaces for hyperparameters. Presently, Hyperopt incorporates three implemented algorithms: random search, a tree of Parzen estimators, and an adaptive tree of Parzen estimators.

5.2.2 Evaluation criteria

A metric is necessary to assess the values of the predicted time series $\hat{y} \in \mathbb{R}^n$, with the true values denoted as $y \in \mathbb{R}^n$. Three metrics are employed to evaluate the predicted values: mean absolute error (MAE), root mean square error (RMSE), and mean absolute percentage error (MAPE). The expressions for each metric are provided in Table 5.2.

Accurate predictions are presumed when the metric values are small. In this experiment, the predictive performance is evaluated using the previously mentioned metrics. As highlighted in [95], MAPE is relatively more stable. Therefore, when the results vary across criteria, we designate MAPE as the benchmark, as suggested in [57].

Table 5.2: Evaluation indices

Metric	expression
MAE	$\frac{1}{N} \sum_{n=1}^N \hat{y}_n - y_n $
RMSE	$\sqrt{\frac{1}{N} \sum_{n=1}^N (\hat{y}_n - y_n)^2}$
MAPE	$\frac{1}{N} \sum_{n=1}^N \left \frac{\hat{y}_n - y_n}{y_n} \right $

5.2.3 Prediction of FTS based on asymmetric extension

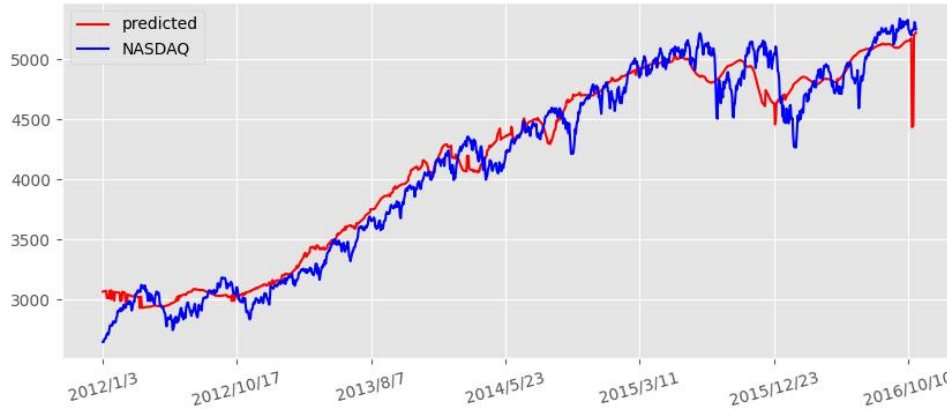
We emphasize that the data utilized for predicting the FTS consists of IMFs derived through IF on the asymmetric extension of the NASDAQ time series, along with the IMFogram obtained from the time-frequency representation of the asymmetric extension of NASDAQ and the IMFs obtained through IF on the asymmetric extension. Table 5.3 presents the performance of the proposed fusion neural networks based on three metrics.

The model comprises a total of 105,657 parameters, with 105,611 being trainable and 46 non-trainable. Figure 5.16 illustrates both the NASDAQ signal and its corresponding prediction. The prediction signal is represented in red, while the NASDAQ signal is depicted in blue.

³Hyperopt github website: <https://github.com/hyperopt/hyperopt>, Nov. 2020.

Table 5.3: The performance of different metric on asymmetric extension of NASDAQ

Training data			Validating data			Testing data		
MAE	RMSE	MAPE	MAE	RMSE	MAPE	MAE	RMSE	MAPE
87.89	106.46	0.25	116.39	155.46	0.024	77.39	138.60	0.014

**Figure 5.16:** NASDAQ and its asymmetric extension signal prediction

5.2.4 Prediction of FTS based on symmetric extension

Table 5.4: The performance of different metric on symmetric extension of NASDAQ

Training data			Validating data			Testing data		
MAE	RMSE	MAPE	MAE	RMSE	MAPE	MAE	RMSE	MAPE
114.51	139.18	0.029	164.79	191.53	0.033	311.67	325.30	0.059

The input data for the proposed fusion neural networks is generated using the symmetric extension of the NASDAQ time series. The IMFs are obtained from the symmetric extension of NASDAQ using IF algorithm. These generated IMFs, along with the symmetric extension of NASDAQ, are utilized to create the IMFogram. Both datasets then serve as inputs for the proposed fusion neural networks. Table 5.4 presents the performance of the proposed fusion neural networks based on three metrics.

The model consists of a total of 52,942 parameters, with 52,912 being trainable and 30 non-trainable. Figure 5.17 illustrates both the NASDAQ signal and its corresponding prediction. The prediction signal is shown in red, while the NASDAQ signal is depicted in blue.

5.2.5 Prediction of FTS based on asymmetric and symmetric extension

The Intrinsic Mode Functions (IMFs) are generated using IF algorithm applied to the asymmetric and symmetric extensions. These obtained IMFs, along with the asymmetric and symmetric extensions, are utilized to create the IMFogram. The resulting two datasets serve as inputs for the proposed fusion neural network, aiming to predict the NASDAQ time series. The predicted NASDAQ time series is represented in red in Figure 5.18, while the original NASDAQ time series

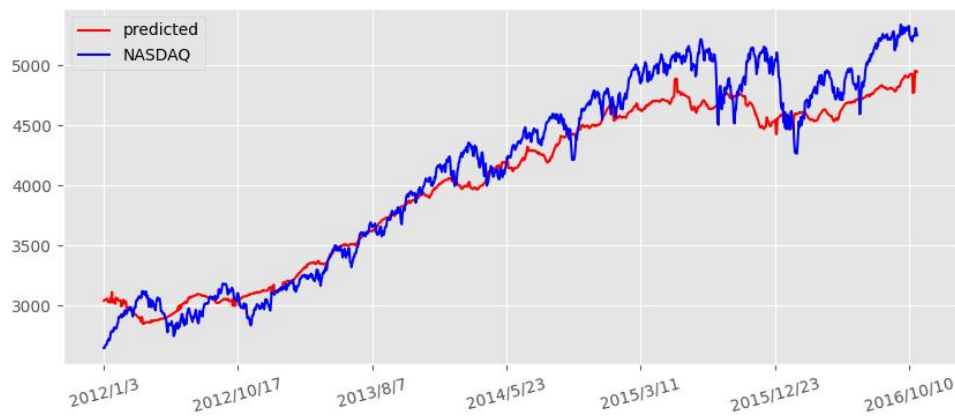


Figure 5.17: NASDAQ and its symmetric extension signal prediction

is depicted in blue.

The model comprises a total of 88,855 parameters, with 88,799 being trainable and 56 non-trainable.

Table 5.5: The performance of different metric on asymmetric and symmetric extension of NASDAQ

Training data			Validating data			Testing data		
MAE	RMSE	MAPE	MAE	RMSE	MAPE	MAE	RMSE	MAPE
69.96	87.32	0.019	90.24	119.44	0.019	105.40	197.99	0.020

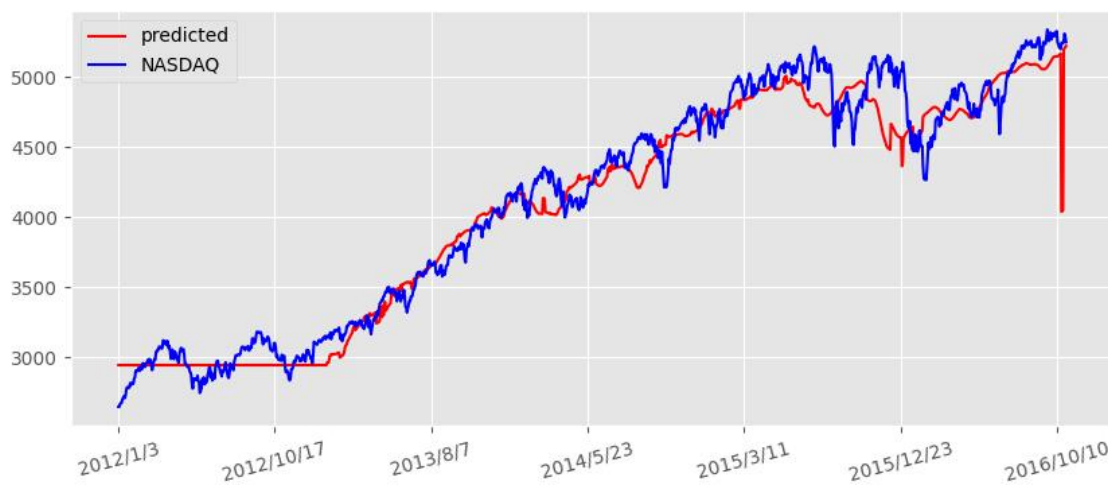


Figure 5.18: NASDAQ and its asymmetric and symmetric extension signal prediction

5.2.6 Prediction of FTS based on symmetric and asymmetric extension

The symmetric and asymmetric extensions are employed to mitigate boundary effects on NASDAQ. The resulting extended signal is subsequently decomposed using the IF algorithm to the IMFs. These extended signals, along with the obtained IMFs, are utilized to create an IMFogram a time-frequency representation of the symmetric and asymmetric extended signal. The two datasets, namely the IMFs and IMFogram, serve as input data for the proposed fusion neural network.

The model consists of a total of 107,776 parameters, with 107,730 being trainable and 46 non-trainable. Figure 5.19 illustrates both the NASDAQ signal and its corresponding prediction, with the prediction signal shown in red and the NASDAQ signal depicted in blue.

Table 5.6: The performance of different metric on symmetric and asymmetric extension of NASDAQ

Training data			Validating data			Testing data		
MAE	RMSE	MAPE	MAE	RMSE	MAPE	MAE	RMSE	MAPE
101.90	123.85	0.026	150.81	176.53	0.030	294.23	318.19	0.056

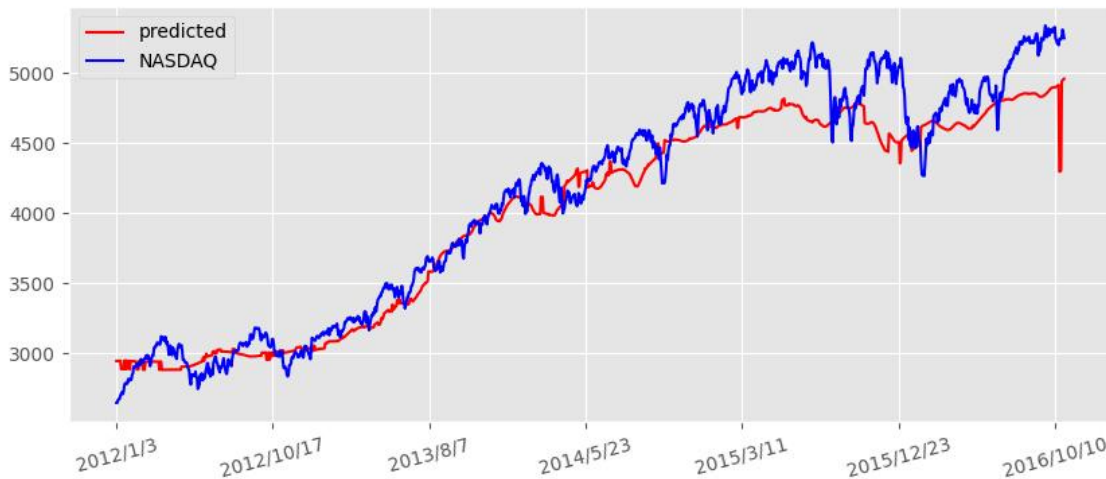


Figure 5.19: NASDAQ and its symmetric and asymmetric extension signal prediction

The best predictions are obtained for metrics with small values, indicating low errors between the predicted values and the true values. It appears that the asymmetric extension predicts better than the other signal extensions across almost all metrics within all subdivisions of the training, validation, and testing datasets. Refer to Table 5.3 and Figure 5.16 for details.

The model behaves quite strangely with both asymmetric and symmetric extensions. While it predicts better than all other extensions on the training and validation sets, it does not generalize well on the testing data, especially concerning the asymmetric extension, and performs poorly on the testing set. The instability in the asymmetric and symmetric extensions is illustrated in Figure 5.18, where we observe that the model predicts a constant value of 2946.001 within the range from 1 to 266.

The symmetric and asymmetric extensions perform better than the symmetric extension. Tables 5.4 and 5.6 illustrate how the symmetric and asymmetric extensions outperform the symmetric extension.

6 Conclusion and future work

Time series analysis has become increasingly prevalent over the last five decades across various domains such as finance, weather, biology, earthquakes, and neuroscience. Time series data originating from these fields tends to exhibit non-stationarity and is often generated by non-linear sources. These characteristics contribute to the classification of time series processing as one of the top 10 challenging problems [59]. Traditional statistical methods face limitations in processing non-linear and non-stationary time series as they predominantly rely on the assumption of stationarity and linearity in the data.

Numerous techniques have been proposed for processing non-stationary and non-linear time series. Machine learning methods, in particular, have demonstrated remarkable results; however, they often lack an explicit mechanism for handling the inherent non-stationarity of time series data. Recent approaches leverage robust features that possess the capability to capture latent information embedded in time series data. However, the utilization of domain-specific features tailored to each field is both time-consuming and costly [59].

This work introduces a novel method for processing time series by integrating the IF and fusion neural network techniques. The fusion neural networks, combining Artificial Neural Networks (ANN) and Convolutional Neural Networks (CNN), are employed for time series prediction. The IF technique is applied to decompose the signal into "quasi-stationary" IMFs. These resulting IMFs, along with the original signal, are utilized to generate an IMFogram. Subsequently, the IMFs serve as input for the ANN, while the IMFogram is used as input for the CNN. The outputs from the ANN and CNN are combined, passed through a dense layer, and subjected to a loss function to measure the error between the predicted and true values.

The proposed framework incorporates the advantages of the IF algorithm, Artificial Neural Network (ANN), and Convolutional Neural Network (CNN) methods, emphasizing the following three points:

1. The framework excels in handling non-stationary signals, thanks to the inherent capability of IF in addressing non-stationarity.
2. The combination of resulting IMFs with the original signal yields an IMFogram. This, along with the IMFs, serves as input for the fusion neural network, providing the framework with the capacity to learn from two distinct data types.
3. The utilization of CNN and ANN enables the proposed framework to adaptively extract deep and global features from time series data.

The proposed framework can be generalized into a universal approach, applicable for extracting features from time series across diverse fields. Its versatility will be demonstrated by evaluating the framework on various datasets exhibiting different instabilities. Additionally, the framework can be adaptable for studying profitability, involving the computation of a single long-short trading strategy to assess its trading performance, accounting for transaction costs or otherwise.

Furthermore, potential enhancements to the proposed framework include the integration of specialized Convolutional Neural Network (CNN) systems to monitor and compare its performance with various existing frameworks.

References

- [1] P. H. Franses and H. Ghijssels, "Additive outliers, garch and forecasting volatility," *International Journal of Forecasting*, vol. 15, no. 1, pp. 1–9, 1999.
- [2] N. Sarantis, "Nonlinearities, cyclical behaviour and predictability in stock markets: international evidence," *International Journal of Forecasting*, vol. 17, no. 3, pp. 459–482, 2001. Reassessing Modern Business Cycles.
- [3] L. Rabiner and B. Juang, "An introduction to hidden markov models," *IEEE ASSP Magazine*, vol. 3, no. 1, pp. 4–16, 1986.
- [4] Y. Chen and Y. Hao, "A feature weighted support vector machine and k-nearest neighbor algorithm for stock market indices prediction," *Expert Systems with Applications*, vol. 80, pp. 340–355, 2017.
- [5] T. Anbalagan and S. U. Maheswari, "Classification and prediction of stock market index based on fuzzy metagraph," *Procedia Computer Science*, vol. 47, pp. 214–221, 2015. Graph Algorithms, High Performance Implementations and Its Applications (ICGHIA 2014).
- [6] R. K. MacKinnon and C. K.-S. Leung, "Stock price prediction in undirected graphs using a structural support vector machine," *2015 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, vol. 1, pp. 548–555, 2015.
- [7] H. Lee, R. Grosse, R. Ranganath, and A. Ng, "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations," in *Proceedings of the 26th International Conference On Machine Learning, ICML 2009*, Proceedings of the 26th International Conference On Machine Learning, ICML 2009, pp. 609–616, 2009. 26th International Conference On Machine Learning, ICML 2009 ; Conference date: 14-06-2009 Through 18-06-2009.
- [8] J.-Z. Wang, J.-J. Wang, Z.-G. Zhang, and S.-P. Guo, "Forecasting stock indices with back propagation neural network," *Expert Systems with Applications*, vol. 38, no. 11, pp. 14346–14355, 2011.
- [9] M. Khashei, M. Bijari, and G. A. Raissi Ardali, "Improvement of auto-regressive integrated moving average models using fuzzy logic and artificial neural networks (annns)," *Neurocomputing*, vol. 72, no. 4, pp. 956–967, 2009. Brain Inspired Cognitive Systems (BICS 2006) / Interplay Between Natural and Artificial Computation (IWINAC 2007).
- [10] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.
- [11] G. Zhang, B. Eddy Patuwo, and M. Y. Hu, "Forecasting with artificial neural networks:: The state of the art," *International Journal of Forecasting*, vol. 14, no. 1, pp. 35–62, 1998.
- [12] C. W. J. Granger and A. Ap, "An introduction to bilinear time series models," 1978.
- [13] H. Tong and K. S. Lim, "Threshold autoregression, limit cycles and cyclical data," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 42, no. 3, pp. 245–268, 1980.
- [14] R. F. Engle, "Autoregressive conditional heteroscedasticity with estimates of the variance of united kingdom inflation," *Econometrica: Journal of the econometric society*, pp. 987–1007, 1982.
- [15] T. Bollerslev, "Generalized autoregressive conditional heteroskedasticity," *Journal of econometrics*, vol. 31, no. 3, pp. 307–327, 1986.
- [16] D. A. Hsieh, "Chaos and nonlinear dynamics: application to financial markets," *The journal of finance*, vol. 46, no. 5, pp. 1839–1877, 1991.

- [17] D. Zhang and L. Zhou, "Discovering golden nuggets: data mining in financial application," *IEEE Trans. Syst. Man Cybern. Part C*, vol. 34, pp. 513–522, 2004.
- [18] S.-H. Chen, *Genetic Algorithms and Genetic Programming in Computational Finance: An Overview of the Book*, pp. 1–26. Boston, MA: Springer US, 2002.
- [19] A. Ponsich, A. L. Jaimes, and C. A. C. Coello, "A survey on multiobjective evolutionary algorithms for the solution of the portfolio optimization problem and other finance and economics applications," *IEEE Transactions on Evolutionary Computation*, vol. 17, no. 3, pp. 321–344, 2013.
- [20] R. Aguilar-Rivera, M. Valenzuela-Rendón, and J. Rodríguez-Ortiz, "Genetic algorithms and darwinian approaches in financial applications: A survey," *Expert Systems with Applications*, vol. 42, no. 21, pp. 7684–7697, 2015.
- [21] Y. Li and W. Ma, "Applications of artificial neural networks in financial economics: A survey," in *2010 International Symposium on Computational Intelligence and Design*, vol. 1, pp. 211–214, 2010.
- [22] M. Tkáč and R. Verner, "Artificial neural networks in business: Two decades of research," *Applied Soft Computing*, vol. 38, pp. 788–804, 2016.
- [23] A. Khadjeh Nassirtoussi, S. Aghabozorgi, T. Ying Wah, and D. C. L. Ngo, "Text mining for market prediction: A systematic review," *Expert Systems with Applications*, vol. 41, no. 16, pp. 7653–7670, 2014.
- [24] C. Kearney and S. Liu, "Textual sentiment in finance: A survey of methods and models," *International Review of Financial Analysis*, vol. 33, no. C, pp. 171–185, 2014.
- [25] B. S. Kumar and V. Ravi, "A survey of the applications of text mining in financial domain," *Knowledge-Based Systems*, vol. 114, pp. 128–147, 2016.
- [26] B. Vanstone and C. Tan, "A survey of the application of soft computing to investment and financial trading," in *Proceedings of the Eighth Australian and New Zealand Intelligent Information Systems Conference (ANZIIS 2003)* (B. Lovell, D. Campbell, C. Fookes, and A. Maeder, eds.), pp. 211–216, The Australian Pattern Recognition Society, 2003. Copyright The Australian Pattern Recognition Society 2003. All rights reserved. Permission granted. ; Australian and New Zealand Intelligent Information Systems Conference, ANZIIS 2003 ; Conference date: 10-12-2003 Through 12-12-2003.
- [27] E. Hajizadeh, H. Davari-Ardakani, and J. Shahrabi, "Application of data mining techniques in stock markets: A survey," *Journal of Economics and International Finance*, vol. 2, pp. 109–118, 08 2010.
- [28] B. Nair and V. Mohandas, "Artificial intelligence applications in financial forecasting-a survey and some empirical results," *Intelligent Decision Technologies*, vol. 9, pp. 99–140, 01 2015.
- [29] R. C. Cavalcante, R. C. Brasileiro, V. L. Souza, J. P. Nobrega, and A. L. Oliveira, "Computational intelligence and financial markets: A survey and future directions," *Expert Systems with Applications*, vol. 55, pp. 194–211, 2016.
- [30] B. Krollner, B. Vanstone, and G. Finnie, "Financial time series forecasting with machine learning techniques: A survey," 01 2010.
- [31] P. Yoo, M. Kim, and T. Jan, "Machine learning techniques and use of event information for stock market prediction: A survey and evaluation," in *International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06)*, vol. 2, pp. 835–841, 2005.

- [32] Y. Bengio, A. Courville, and P. Vincent, “Representation learning: A review and new perspectives,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.
- [33] G. E. Hinton, S. Osindero, and Y.-W. Teh, “A Fast Learning Algorithm for Deep Belief Nets,” *Neural Computation*, vol. 18, pp. 1527–1554, 07 2006.
- [34] H. Saleh, *The Deep Learning with PyTorch Workshop*. July 2020.
- [35] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” *CoRR*, vol. abs/1512.03385, 2015.
- [36] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems* (F. Pereira, C. Burges, L. Bottou, and K. Weinberger, eds.), vol. 25, Curran Associates, Inc., 2012.
- [37] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–9, 2015.
- [38] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen, J. Chen, J. Chen, Z. Chen, M. Chrzanowski, A. Coates, G. Diamos, K. Ding, N. Du, E. Elsen, J. Engel, W. Fang, L. Fan, C. Fougner, L. Gao, C. Gong, A. Hannun, T. Han, L. Johannes, B. Jiang, C. Ju, B. Jun, P. LeGresley, L. Lin, J. Liu, Y. Liu, W. Li, X. Li, D. Ma, S. Narang, A. Ng, S. Ozair, Y. Peng, R. Prenger, S. Qian, Z. Quan, J. Raiman, V. Rao, S. Satheesh, D. Seetapun, S. Sengupta, K. Srinet, A. Sriram, H. Tang, L. Tang, C. Wang, J. Wang, K. Wang, Y. Wang, Z. Wang, Z. Wang, S. Wu, L. Wei, B. Xiao, W. Xie, Y. Xie, D. Yogatama, B. Yuan, J. Zhan, and Z. Zhu, “Deep speech 2 : End-to-end speech recognition in english and mandarin,” in *Proceedings of The 33rd International Conference on Machine Learning* (M. F. Balcan and K. Q. Weinberger, eds.), vol. 48 of *Proceedings of Machine Learning Research*, (New York, New York, USA), pp. 173–182, PMLR, 20–22 Jun 2016.
- [39] J. Hirschberg and C. D. Manning, “Advances in natural language processing,” *Science*, vol. 349, no. 6245, pp. 261–266, 2015.
- [40] J. Egger, C. Gsaxner, A. Pepe, K. L. Pomykala, F. Jonske, M. Kurz, J. Li, and J. Kleesiek, “Medical deep learning—a systematic meta-review,” *Computer Methods and Programs in Biomedicine*, vol. 221, p. 106874, 2022.
- [41] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, pp. 436–44, 05 2015.
- [42] A. M. O. Omer Berat Sezer, M. Ugur Gudelek, “Financial time series forecasting with deep learning : A systematic literature review: 2005-2019,” *arXiv:1911.13288v1*, 11 2019.
- [43] J. Schmidhuber, “Deep learning in neural networks: An overview,” *Neural Networks*, vol. 61, pp. 85–117, 2015.
- [44] L. Deng and D. Yu, “Deep learning: Methods and applications,” *Foundations and Trends® in Signal Processing*, vol. 7, no. 3–4, pp. 197–387, 2014.
- [45] M. Gardner and S. Dorling, “Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences,” *Atmospheric Environment*, vol. 32, no. 14, pp. 2627–2636, 1998.
- [46] R. Pascanu, T. Mikolov, and Y. Bengio, “On the difficulty of training recurrent neural networks,” in *Proceedings of the 30th International Conference on Machine Learning* (S. Dasgupta and D. McAllester, eds.), vol. 28 of *Proceedings of Machine Learning Research*, (Atlanta, Georgia, USA), pp. 1310–1318, PMLR, 17–19 Jun 2013.

- [47] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, and Johnson, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv*, 2016.
- [48] G. Van Houdt, C. Mosquera, and G. Nápoles, "A review on the long short-term memory model," *Artificial Intelligence Review*, vol. 53, 12 2020.
- [49] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in Neural Information Processing Systems* (Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, eds.), vol. 27, Curran Associates, Inc., 2014.
- [50] C. A. Ronao and S.-B. Cho, "Human activity recognition with smartphone sensors using deep learning neural networks," *Expert Systems with Applications*, vol. 59, pp. 235–244, 2016.
- [51] T. Kolarik and G. Rudorfer, "Time series forecasting using neural networks," in *APL Conference*, 1994.
- [52] J. Yao, C. L. Tan, and H.-L. Poh, "Neural networks for technical analysis: A study on klci," *International Journal of Theoretical and Applied Finance*, vol. 02, pp. 221–241, 1999.
- [53] G. Zhang, "Time series forecasting using a hybrid arima and neural network model," *Neurocomputing*, vol. 50, pp. 159–175, 2003.
- [54] T. A. E. Ferreira, G. C. Vasconcelos, and P. J. L. Adeodato, "A new intelligent system methodology for time series forecasting with artificial neural networks," *Neural Processing Letters*, vol. 28, pp. 113–129, 2008.
- [55] M. R. Hassan, B. Nath, and M. Kirley, "A fusion model of hmm, ann and ga for stock market forecasting," *Expert Systems with Applications*, vol. 33, no. 1, pp. 171–180, 2007.
- [56] X. Zhang, M. Chen, M. Wang, Y. Ge, and H. Stanley, "A novel hybrid approach to baltic dry index forecasting based on a combined dynamic fluctuation network and artificial intelligence method," *Applied Mathematics and Computation*, vol. 361, pp. 499–516, 2019.
- [57] F. Zhou, H. min Zhou, Z. Yang, and L. Yang, "Emd2fnn: A strategy combining empirical mode decomposition and factorization machine based neural network for stock market trend prediction," *Expert Systems with Applications*, vol. 115, pp. 136–151, 2019.
- [58] F. Zhou, H. Zhou, Z.-H. Yang, and L.-H. Yang, "A 2-stage strategy for non-stationary signal prediction and recovery using iterative filtering and neural network," *Journal of Computer Science and Technology*, vol. 34, pp. 318–338, 03 2019.
- [59] F. Zhou, H. Zhou, Z. Yang, and L. Gu, "If2cnn: Towards non-stationary time series feature extraction by integrating iterative filtering and convolutional neural networks," *Expert Systems with Applications*, vol. 170, p. 114527, 2021.
- [60] Y. Tang, Z. Song, Y. Zhu, H. Yuan, M. Hou, J. Ji, C. Tang, and J. Li, "A survey on machine learning models for financial time series forecasting," *Neurocomputing*, vol. 512, pp. 363–380, 2022.
- [61] S. McDonald, S. Coleman, T. McGinnity, Y. Li, and A. Belatreche, "A comparison of forecasting approaches for capital markets," in *Unknown Host Publication*, (United States), pp. 32–39, IEEE, Mar. 2014. IEEE Computational Intelligence for Financial Engineering and Economics ; Conference date: 27-03-2014.
- [62] R. S. TSAY, *Analysis of Financial Time Series*. Chicago: A JOHN WILEY, SONS, INC., PUBLICATION, 2005.
- [63] A. Jablonski and K. Dziedziech, "Intelligent spectrogram – a tool for analysis of complex non-stationary signals," *Mechanical Systems and Signal Processing*, vol. 167, p. 108554, 2022.

- [64] A. Papandreou-Suppappola, *Applications in time-frequency signal processing*. CRC press, 2018.
- [65] J. A. Gallego, E. Rocon, J. O. Roa, J. C. Moreno, and J. L. Pons, “Real-time estimation of pathological tremor parameters from gyroscope data,” *Sensors*, vol. 10, no. 3, pp. 2129–2149, 2010.
- [66] G. Shafiq, S. Tatinati, W. T. Ang, and K. C. Veluvolu, “Automatic identification of systolic time intervals in seismocardiogram,” *Scientific reports*, vol. 6, no. 1, p. 37524, 2016.
- [67] G. Shafiq and K. C. Veluvolu, “Surface chest motion decomposition for cardiovascular monitoring,” *Scientific reports*, vol. 4, no. 1, p. 5093, 2014.
- [68] K. C. Veluvolu and W. T. Ang, “Estimation of physiological tremor from accelerometers for real-time applications,” *Sensors*, vol. 11, no. 3, pp. 3020–3036, 2011.
- [69] M. Tarvainen, S. Georgiadis, J. Lipponen, M. Hakkarainen, and P. Karjalainen, “Time-varying spectrum estimation of heart rate variability signals with kalman smoother algorithm,” in *2009 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pp. 1–4, IEEE, 2009.
- [70] Y. Wang and K. C. Veluvolu, “Time-frequency analysis of non-stationary biological signals with sparse linear regression based fourier linear combiner,” *Sensors*, vol. 17, no. 6, 2017.
- [71] N. E. Huang, Z. Shen, S. R. Long, M. C. Wu, H. H. Shih, Q. Zheng, N.-C. Yen, C. C. Tung, and H. H. Liu, “The empirical mode decomposition and the hilbert spectrum for nonlinear and non-stationary time series analysis,” *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, vol. 454, pp. 903–995, mar 1998.
- [72] Z. Wu and N. Huang, “Ensemble empirical mode decomposition: a noise-assisted data analysis method,” *Advances in Adaptive Data Analysis*, vol. 1, pp. 1–41, 01 2009.
- [73] L. Lin, Y. Wang, and H. Zhou, “Iterative filtering as an alternative algorithm for empirical mode decomposition,” *Advances in Adaptive Data Analysis*, vol. 1, pp. 543–560, 10 2009.
- [74] G. Thakur, E. Brevdo, N. S. Fućkar, and H.-T. Wu, “The synchrosqueezing algorithm for time-varying spectral analysis: Robustness properties and new paleoclimate applications,” *Signal processing*, vol. 93, no. 5, pp. 1079–1094, 2013.
- [75] K. Dragomiretskiy and D. Zosso, “Variational mode decomposition,” *IEEE transactions on signal processing*, vol. 62, no. 3, pp. 531–544, 2013.
- [76] S. Chen, X. Dong, Z. Peng, W. Zhang, and G. Meng, “Nonlinear chirp mode decomposition: A variational method,” *IEEE Transactions on Signal Processing*, vol. 65, no. 22, pp. 6024–6037, 2017.
- [77] T. Eriksen and N. ur Rehman, “Data-driven signal decomposition approaches: A comparative analysis,” 2022.
- [78] J. Harmouche, D. Fourer, F. Auger, P. Borgnat, and P. Flandrin, “The sliding singular spectrum analysis: A data-driven nonstationary signal decomposition tool,” *IEEE Transactions on Signal Processing*, vol. 66, no. 1, pp. 251–263, 2017.
- [79] A. Cicone, J. Liu, and H. Zhou, “Adaptive local iterative filtering for signal decomposition and instantaneous frequency analysis,” *Applied and Computational Harmonic Analysis*, vol. 41, no. 2, pp. 384–411, 2016. Sparse Representations with Applications in Imaging Science, Data Analysis, and Beyond, Part II.
- [80] A. Cicone and H. Zhou, “Multidimensional iterative filtering method for the decomposition of high-dimensional non-stationary signals,” *Numerical Mathematics: Theory, Methods and Applications*, vol. 10, no. 2, p. 278–298, 2017.

- [81] L. Lin, Y. Wang, and H. Zhou, “Iterative filtering as an alternative algorithm for empirical mode decomposition,” *Adv. Data Sci. Adapt. Anal.*, vol. 1, pp. 543–560, 2009.
- [82] P. Barbe, A. Cicone, W. S. Li, and H. Zhou, “Time-frequency representation of nonstationary signals: the imfogram,” 2021.
- [83] A. Cicone, W. S. Li, and H. Zhou, “New theoretical insights in the decomposition and time-frequency representation of nonstationary signals: the imfogram algorithm,” 2022.
- [84] Y. Wang and Z. Zhou, “On the convergence of iterative filtering empirical mode decomposition,” *Excursions in Harmonic Anal.*, vol. 2, 11 2013.
- [85] P. Petersen and F. Voigtlaender, “Optimal approximation of piecewise smooth functions using deep relu neural networks,” *Neural Networks*, vol. 108, pp. 296–330, 2018.
- [86] C. Marcati, J. A. A. Opschoor, P. C. Petersen, and C. Schwab, “Exponential ReLU neural network approximation rates for point and edge singularities,” *Foundations of Computational Mathematics*, jun 2022.
- [87] S. Thomas, *PyTorch Deep Learning Hands-On*. Packt Publishing Ltd, April 2019.
- [88] S. R. Dubey, S. K. Singh, and B. B. Chaudhuri, “Activation functions in deep learning: A comprehensive survey and benchmark,” *Neurocomputing*, vol. 503, pp. 92–108, 2022.
- [89] K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification,” in *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 1026–1034, 2015.
- [90] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” 2015.
- [91] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014.
- [92] P. Rigollet and J. Hütter, “Lecture notes on high-dimensional statistics,” 2015.
- [93] A. Stallone, A. Cicone, and M. Materassi, “New insights and best practices for the successful use of empirical mode decomposition, iterative filtering and derived algorithms,” *Scientific reports*, vol. 10, p. 15161, 09 2020.
- [94] M. K. P. Ng, R. H. Chan, and W.-C. Tang, “A fast algorithm for deblurring models with neumann boundary conditions,” *SIAM J. Sci. Comput.*, vol. 21, pp. 851–866, 1999.
- [95] S. Makridakis, “Accuracy measures: theoretical and practical concerns,” *International Journal of Forecasting*, vol. 9, no. 4, pp. 527–529, 1993.