Università degli Studi di Cagliari

PhD DEGREE

Mathematics and Computer Science

Cycle XXXIV

TITLE OF THE PhD THESIS

Artificial Intelligence Approaches Applied To The Financial Forecasting Domain

Scientific and Disciplinary Sector(s)

S.S.D. INF/01

PhD Student:	Andrea Corriga
Supervisor	Salvatore Carta, Diego Reforgiato Recupero

Final exam. Academic Year 2020/2021
Thesis defence: February 2022 Session

Abstract

Nowadays, Financial Markets represent a crucial part of the world economy. Financial Markets have grown exponentially in the last decades and they provide finance for companies that can be used to hire, invest and grow. On the other hand, Financial Markets provide a big opportunity for people to invest their money in shares or equities to build up money for their future. The possibility to increase one's capital through investment has led researchers in the last decade to focus their work on predicting the performance of the market by exploiting novel Machine Learning and Deep Learning tools and techniques.

Several approaches have been proposed in the literature, ranging from time-series pattern recognition analysis to more complex approaches based on Machine Learning and Deep Learning.

Following this trend, the main contribution of this dissertation is the proposal of three novel approaches to tackle these issues. Firstly, starting from scratch, we propose a fully automated optimized ensemble approach, where an optimized feature selection process has been combined with an automatic ensemble Machine Learning strategy. Secondly, we exploit a Deep Learning approach with an ensemble of CNNs, trained over Gramian Angular Fields (GAF) images, generated from time series related to the Standard & Poor's 500 index Future. More precisely, a multi-resolution imaging approach is used to feed each CNN, enabling the analysis of different time intervals for a single observation. Finally, we propose an improvement of the previous approach with a multi-layer and multi-ensemble stock trader. This method starts by pre-processing data with hundreds of Deep Neural Networks and then, a reward-based classifier is used to maximize profit and generate stock signals through different iterations.

At the end of this dissertation, accompanying the current Machine Learning Interpretability trend, we propose a Visual Framework for in-depth analysis of the results obtained from Deep Learning approaches, tackling classification tasks within the financial domain and aiming at a better interpretation and explanation of the trained Deep Learning models. The proposed Framework offers a modular view, both general and targeted, of results data, providing several financial-specific metrics.

Acknowledgements

As one said: *part of the journey is the end* and this thesis ends my long education journey. At the end of high school, I didn't know if I wanted to start college but now, looking back, I feel like I have a lot of people to thank.

First of all, I would like to express my deepest gratitude to my two supervisors Prof. Salvatore Mario Carta and Prof. Diego Reforgiato Recupero for guiding me on this educational and personal growth path. In particular I would like to thank Prof. Salvatore who has been an unstoppable mentor who cared so much about me, my work, and who responded to my questions and troubles so promptly; your hardness will be my strength tomorrow.

Secondly, I would like to thank Alessandro Sebastian Podda with whom I spent great time during my PhD. You have been a source of inspiration, entertainment, and great discussions.

A special thanks to Enrico Podda, an invaluable and irreplaceable friend and office mate. You have always been there when needed.

Outside from the work, I have a lot of people to thank.
I would like to start with my parents Rafaele e Giuliana, because without you and your constant support none of this would have happened. Thanks to my brother Emanuele who, among many other things, gave me a fantastic and super-sweet nephew Eleonora.

My girlfriend Daniela who spent these 3 years of PhD living with me, supporting me in all the difficult moments and being patient and tolerant during the lockdown period.

Finally, I would like to thank all my friends and people who have been part of my life and supported me during my PhD experience. I can't mention you all here but I carry you in my heart.

Furthermore, I have to acknowledge the financial support I have received during my Ph.D. All my research activities have been funded by:

- POR FESR SARDEGNA 2014-2020 2° CALL ALMOSTANORACLE-ATI DMI-VISIOSCIENTIAE – CUP: F21G180002300081;
- POR FESR SARDEGNA 2014-2020 CLUSTER - DOUTDES - Progetto di ricerca dell'Università di Cagliari finanziato dalla R.A.S. CUP: F21B17000860005;
- POR FESR SARDEGNA 2014-2020 2° CALL "SPRINT"- ATI DMI-LA SIA SRL – CUP F21G18000240009;
- PhilHumans – H2020 MSCA EID - European Industrial Doctorates - CUP: F26C18000420006;
- POR FESR SARDEGNA 2014 - 2020 CLUSTER - SARDCOIN - Progetto di ricerca dell'Università di Cagliari finanziato dalla R.A.S., CUP: F21B17000850005.

Statement of Authorship

The thesis entitled *Artificial Intelligence Approaches Applied To Financial Forecasting Domain* and the presented works have been done during my candidature for this PhD degree. The presented novel approaches were carried out by myself, under the coordination of my supervisors and the collaboration of the authors indicated in the respective published papers. During the elaboration of this thesis I have follow appropriate ethics guidelines to conduct this research, I have acknowledged all main sources of help and, when I consulted or quote the work published by others, the source is always given.

Nomenclature

Abbreviations

AI	Artifcial Intelligence
AL	Attention Layer
ANN	Artificial Neural Network
API	Application Programming Interface
ARIMA	Auto Regressive Integrated Moving Average
ASR	Annualized Sharpe Ratio
BN	Batch Normalization
BSE	Bombay Stock Exchange
BP	Back Propagation
CNN	Convolutional Neural Network
CSV	Comma Separated Value
CV	Computer Vision
DL	Deep Learning
DNM	Dendritic Neuron Model
DNN	Deep Neural Network
DRL	Deep Reinforcement Learning
DT	Decision Trees
FTSE	Financial Times-Stock Exchange
FA	Fundamental Analysis
FM	Financial Markets
GPU	Graphic Processing Unit
ICS	Internal covariate shift
LSSVR	Least Squares Support Vector Regression
LSTM	Long Short-Term Memory
MDD	Maximum DrawDown
ML	Machine Learning
MSE	Mean Squared Error
RBF	Radial Basis Function
RF	Random Forest
RNN	Recurrent Neural Network
RoMaD	Return Over Maximum Drawdown
RL	Reinforcement Learning

SGD	Stochastic Gradient Descent
SP500	Standard & Poor 500
SVM	Support Vector Machine
SVR	Support Vector Regressor
TA	Technical Analysis
TOPIX	Tokyo Stock Exchange Price Indexes
WFO	Walk Forward Optimization

Latin Expressions

i.e.	id est, ‘that is’
e.g.	exempli gratia, ’for the sake of example’
ergo	’therefore’

Publications

The research reported in this thesis has contributed to the following publications:

- [1] Carta, S., Corriga, A., Ferreira, A., Reforgiato Recupero, D., Saia, R. A Holistic Auto-Configurable Ensemble Machine Learning Strategy for Financial Trading (2019) Computation, 7 (4), pp. 1-25.
- [2] Barra, S., Carta, S.M., Corriga, A., Podda, A.S., Reforgiato Recupero, D.. Deep Learning and Time Series-to-Image Encoding for Financial Forecasting (2020) IEEE/CAA Journal of Automatica Sinica, 7 (3), art. no. 9080613, pp. 683-692.
- [3] Carta, S., Consoli, S., Corriga, A., Dapiaggi, R., Podda, A.S., Reforgiato Recupero, D. HawkEye: a Visual Framework for Agile Cross-Validation of Deep Learning Approaches in Financial Forecasting (2020) In The 4th International Conference on Future Networks and Distributed Systems (ICFNDS) (pp. 1-6).
- [4] Carta, S., Corriga, A., Ferreira, A., Podda, A.S., Reforgiato Recupero, D. A Multi-Layer and Multi-Ensemble Stock Trader Using Deep Learning and Deep Reinforcement Learning (2021) Applied Intelligence, 51 (2), pp. 889-905.

List of Figures

2.1 Example of hierarchy in Artificial Intelligence field 12

2.2 Machine Learning pipeline process. 13

2.3 Example of clustering algorithms 16

2.4 An example of a Feed-forward Network composed by three hidden layers. 18

2.5 An example of a Convolutional Neural Network. 19

2.6 ML Evaluation techniques: held-out (top), k-fold (middle) walk-forward (bottom) 24

4.1 The proposed two-Step auto adjustable parameters ensemble for market forecasting. This approach optimizes two sets of parameters for a final ensemble to trade in any market. Firstly, in an in-sample dataset with late past data, we optimize the hyper-parameters, considering performance metrics in the testing part of data. Then, these hyper-parameters are transferred to create ensembles for an early past (out of sample) dataset. The individual classifiers have their intrinsic-parameters updated in the validation part of this recent data and then the final ensemble is built. 39

4.2 $Equity\ Curves for SP500\&DAX$ 54

4.3 $Equity\ Curves for FIB\&CL$ 55

5.1 The figure shows the process leading to the generation of the Gramian Angular Fields images: from left to right, the data are first plotted and then the coordinate system is transformed to a polar plane; finally, the GADF and GASF images are generated according to the function defined in Equation 6.3. . 61

5.2 The policy of composition of the multi-resolution GADF images is shown above; in particular, (a), (b), (c) and (d) refer to the same label, but the observations considered in each are aggregated in four different ways. 62

5.3 On the leftmost side of the figure the architecture of the Convolutional Neural Network is shown. On the rightmost side, the overall process of the proposed trading system is depicted. 63
5.4 The net profit obtained by the application of the Buy-and-Hold strategy is shown over the period which has been taken into account for the experiments (https://www.investing.com/indices/us-spx-500-futures). 67
5.5 The figure shows the comparison among the tested approaches. In particular, the blue, orange and green lines represent, respectively, the results obtained by applying our approach, a random guessing approach and a 1D-CNN. The red line evidences the results which are obtained by the Buy-and-Hold strategy. In each plot, the x-axis indicates the ensemble threshold considered, while in the y-axis it is shown the obtained Net Profit, the Romad value, the Annualized Sharpe Ratio and the Sortino Ratio, respectively, from the upper-leftmost image to the bottom-rightmost one. 69
5.6 The above plot shows the comparison, in terms of cumulative profit, between the proposed approach (in blue) and the approach in [5] (in black). Finally, in orange, we show the Buy&Hold benchmark as a benchmark. 72

6.1 The proposed three layered multi-ensemble approach. The first layer stacks decisions from CNNs, which are then used as observations (or states) in a reward-based meta-learner (second layer) in different training iterations. Finally, a last layer fuses the final decisions in a final trading signal. 76
6.2 The VGG-based CNN used to pre-process stock market time series to image data. 78
6.3 The Neural Network architecture of our proposed multi-DQN agents for stock trading. The activation of the first layer is chosen through optimization experiments. 81
6.4 Fusion Layer pipeline of our proposed multi-layer agent. Multiple agents trained after multiple series of different iterations with the environment perform intra-day stock trading, done by choosing among different combinations of actions. The final action to take is decided by majority voting of decisions. 82
6.5 Example of real-world trading simulation through MultiCharts. 86
6.6 Equity curve of our RL-ensemble approach, against the best non-RL baseline (majority voting ensemble) and the Buy-and-Hold strategy. 91

7.1 The high-level architecture of our framework. 96
7.2 Main dashboard of the VT. It features: (1) the toolbar; (2) the summary box; and, (3) the plot area. 98
7.3 Different display modes of the *plot area*: full epoch range (on the left), custom range selection (on the upper right) and single epoch zoom (on the lower right). 99
7.4 The sub-period filter. 99

List of Tables

4.1 Non-anchored and Anchored WFO 41

4.2 Intra-parameters grid 41

4.3 Hyper-parameters Grid 42

4.4 Futures Market Datasets 47

4.5 Time Consumption 50

4.6 Buy and Hold and Single Predictors Performance with Default Configurations 51

4.7 Single Predictors Performance After a Tuning Process Optimized by Accuracy 53

4.8 Ensemble Predictors Performance After a Tuning Process Optimized by Accuracy 53

4.9 Comparison of the proposed approach against the Buy and Hold benchmark by considering different performance metrics in building the proposed self-configurable ensemble. 57

5.1 In the table, for each initialization method, the configuration settings are reported. The "*seed*" column indicates the seed number applied for generating the random weights, according to the used initialization function. The first 10 networks have the "*seed*" set to "*None*", meaning that randomised numbers differ in each execution. The other 10 networks have a fixed "*seed*" meaning that randomised numbers are generated "*seed*". For all configurations, Adam has been used as an optimiser. 64

5.2 The eleven walks used for training, validation and testing the models and how they are composed. 67

5.3 The table shows the quantitative results of the proposed approach. For each metric, the best result is highlighted in bold. 71

6.1 Dataset structure (1-hour time resolution); prices are expressed in market points. 84

6.2 Top-10 results, in terms of Romad, of the parameter optimization experiment performed over the validation set, by considering different metalearner optimizers, layer activations and different levels of explorations in the metalearner policy. Classification metrics are percentages in $[0, 1]$ range, while trading metrics are shown in market points (1 point = 50 USD). . . . 88
6.3 Summary of the comparison between our RL layers and the non-RL ensembling techniques outlined in Sec. 6.2.2, in an out-of-sample trading scenario (September 2018 to August 2019). Best results per trading metric are underlined. Trading metrics are expressed in market points (1 point = 50 USD) and classification metrics are in the interval [0,1]. 89
6.4 Comparison of the real-world trading performance, in terms of annualized returns, between our method and some literature competitors when considering different periods and markets. Our method clearly outperforms all the competitors in the considered settings. 92

Contents

Abstract **i**

Acknowledgements **iii**

Statement of Authorship **v**

Publications **ix**

List of Figures **xi**

List of Tables **xv**

1 Introduction **3**
- 1.1 Context and Research Contributions 3
- 1.2 Dissertation Structure 8

2 Technical background **11**
- 2.1 History of Artificial Intelligence 11
- 2.2 Supervised approaches 13
- 2.3 Unsupervised approaches 16
- 2.4 Deep Learning 17
- 2.5 Experimental Workflow 21
- 2.6 Evaluation Metrics 24
  - 2.6.1 AI Metrics 25
  - 2.6.2 Financial Metrics 26
- 2.7 Software Tools and Technologies 28

3 State of the art **31**

4 A Holistic Auto-Configurable Ensemble Machine Learning Strategy for Financial Trading **35**
- 4.1 Proposed Approach 36
  - 4.1.1 Feature Selection 37

4.1.2 Two-Step Auto Adjustable Parameters Ensemble Creation . . . 38
4.1.3 Policy for Trading . . . 43
4.2 Experimental Setup . . . 46
4.2.1 Formal Notation . . . 46
4.2.2 Datasets . . . 47
4.2.3 Technical Details . . . 49
4.3 Experiments . . . 50
4.3.1 Trading Results . . . 50
4.3.2 Performance Metrics Trade Impact . . . 53

5 Deep Learning and Time Series-to-Image Encoding 59
5.1 The proposed approach . . . 59
5.1.1 Trading strategy . . . 59
5.1.2 Gramian Angular Fields imaging . . . 60
5.1.3 Multi-Resolution Time Series Imaging . . . 61
5.1.4 Ensemble of CNNs . . . 62
5.2 Experimental Settings . . . 64
5.2.1 Evaluation and S&P500 . . . 65
5.2.2 Walks' Definition . . . 66
5.2.3 CNN training and Ensemble policy . . . 68
5.3 Results and Discussion . . . 68

6 A Multi-Layer and Multi-Ensemble Stock Trader Using Deep Learning and Deep Reinforcement Learning 75
6.1 Proposed Approach . . . 75
6.1.1 Layer #1: Stacking trading signals with Convolutional Neural Networks . . . 76
6.1.2 Layer #2: Reinforcement MetaLearner . . . 78
6.1.3 Layer #3: Ensembling Multiple Learners . . . 82
6.2 Experimental Setup . . . 83
6.2.1 Datasets . . . 83
6.2.2 Benchmarks . . . 84
6.2.3 Implementation Details of the Proposed Approach . . . 86
6.3 Experiments . . . 87
6.3.1 Metalearner Parameters Optimization . . . 87
6.3.2 Experimental Results and Comparison With Baselines 89

7 Hawkeye: a Visual Framework for Agile Cross-Validation of Deep Learning Approaches in Financial Forecasting 95
7.1 Introduction . . . 95
7.2 The Proposed Framework . . . 96
7.2.1 Data Organization & Pre-processing . . . 96

7.2.2 Visual Tool 97

8 Conclusions 101

8.1 Future Research Directions 103

Bibliography 105

Chapter 1

Introduction

1.1 Context and Research Contributions

Nowadays, Financial Markets represent the backbone of the modern societies, as the world economy is closely related to their behavior [6]. In this context, the investors play a main role, since their decisions drive the Financial Markets.

In fact, since the dawn of the Financial Markets, people have been trying to build tools able to provide insights and information about the stock price variations in the near future, so to increase the possibilities to invest on the right company [7], future, etc. Since then, the market has become much bigger, and the available instruments for Financial Forecasting have reached an unprecedented efficiency. The literature reports two main economic methodologies largely used to analyze and predict the behavior of Financial Markets. The first is based on *fundamental analysis*, which takes into account the economic elements that may affect the market activities. The second is based on *technical analysis* [8], which takes into account the historical behavior of the market prices. Moreover, the *technical analysis* considers the financial asset behavior as a time series and it is based on the consideration that some behaviors tend to occur again in the future [9, 10].

Fundamental Analysis. FA is a long-term oriented exercise and practice. FA of stocks determines the fundamental value of a stock by analyzing available information with a special emphasis on accounting information. FA of stocks proceeds in two steps (i) the first step inspects the financial data of a corporation its profit-and-loss account and its balance sheet and aims at assessing future earnings (ii) the second step traces the causal link from future earnings to market value. [11] In other words FA is a method of measuring a security's intrinsic value by examining related economic and financial factors. FA study anything that can affect the security's value, from macroeconomic factors such as the state of the economy and industry conditions to microeconomic factors like the effectiveness of the company's management.

Technical Analysis. TA is the study of past price movements with the goal to predict future price movements from the past. The philosophy behind TA is that information is gradually discounted in the price of an asset. Except for a crash once in a while there is no 'big bang' price movement that immediately discounts all available information. It is said that price gradually moves to new highs or new lows and that trading volume goes with the prevailing trend. Therefore most popular technical trading rules are trend following techniques such as moving averages and filters. TA tries to detect changes in investors' sentiments in an early stage and tries to profit from them. It is said that these changes in sentiments cause certain patterns to occur repeatedly in the price charts, because people react in the same way in equal circumstances. [12]

Differently from the past, Financial Markets have grown exponentially in recent decades and, at the same time, more financial instruments have been introduced to predict their behavior. There are several information and communication technologies that have been employed within the financial domain, so investors are now supported by many Artificial Intelligence instruments that help them to take decisions. Such instruments can exploit a diverse number of techniques [13], from simple statistical approaches to those more sophisticated based on Deep Learning, Social Media Analysis, Natural Language Processing, Sentiment Analysis and so on [14, 15, 16, 17, 18, 19, 20, 21].

Machine Learning solutions have been widely adopted in the context of financial time series forecasting. They usually operate by using a *supervised* strategy, where classifiers (*e.g.*, *Naive Bayes*, *Decision Trees*, *Support Vector Machines*, *etc.*) label the data in order to learn their behavior and classify new data into a number of classes (*i.e.*, in the stock market, such classes can be considered as prices going up and down). Deep Learning approaches have also been proposed in the literature, where a Deep Convolutional Neural Network can be used to perform *classification* or *regression* task, which means predicting the daily direction (positive or negative) of the market for classifications whereas predicting the daily expected price for *regressions*. A common classification task within the financial domain consists of defining an *intraday trading* strategy, which targets three possible actions to perform daily:

- a `long` action, which consists of buying the stock when the market opens in the next day, and then selling it before the market closes;

- a `short` action, which consists of selling the stock (using the mechanism of the short sales) when the market opens, and then buying it before the market closes;
- a `flat` action, which consists of deciding not to invest in that day.

In recent years, Machine Learning approaches have been validated to perform stock market predictions on the basis of historical time series. The research performed in recent years in this field is growing both in terms of literature production and in tools generation [22, 23]; also, an increasing number of studies involves the use of traditional Machine Learning approaches and Neural Network models.

The computer-aided stock trading is basically composed of two steps: (i) analysis of past market behavior, and (ii) taking the optimal stock trading decision. To perform such tasks, time-series data from past prices are usually considered as input. These data are usually given by the market under different resolutions (minutes, hours, days, etc) and contain information such as open prices, close prices, among others.

Indeed, since input data are structured as *time-series*, the chronological order of the samples must be taken into account during the training process, making the model validation process more difficult. For this reason, in the Financial Forecasting domain, methods based on *walk-forward* validation, such as rolling window or expanding window approaches, are preferred [24]. Here, input data are partitioned into consecutive time windows (or *walks*), each composed by three different chunks: the first (largest) portion, the *training set* is used to train the classifier; the second slice or *validation set* is used to perform parameters optimization and evaluate the robustness of the model; and the last and remaining part (*test set*) is reserved for back-testing. Following this approach, each walk is then defined by shifting forward the time window of its predecessor (by a period equivalent to the test set time interval), and the process is iterated. However, this family of validation methods, although effective, introduces an additional crucial aspect to examine when analysing results in this domain: the choice on how to temporally partition the three different sets for training, validation and test.

There are also methods known in statistical analysis that perform regression, which consists of a set of statistical processes for estimating the relationships among variables [25], with the goal of predicting the exact stock price for a day. Although both technical and fundamental data can be used as input data to Machine Learning approaches, fundamental analysis data do not allow a reliable and high frequency trading for two reasons: (i) these type of information are published at periodic times (*e.g.*, every trimester); and (ii) they are responsibility of companies, so they can be liable to frauds. Therefore, most of Machine Learning approaches do not rely on fundamental information, using diverse other information from technical analysis such as lagged prices, returns, technical indicators and even news. One more difference between technical and fundamental analysis is that the latter might often use sensitive data (e.g. revenue of companies) and policy procedures should be defined to guarantee privacy, protection and not disclosure of the data.

Therefore, the research in this area is one of the most active amongst the pattern recognition related topics, and at the same time it is one of the most challenging. This is mainly due to the fact that stock prices are often influenced by factors which are quite hard predictable like political events, the behaviour of the other stock markets, the psychology of the investors [26], wars and many other events that rarely happen as the COVID-19 pandemic [27, 28]; these aspects tend to model the market as an entity which is *dynamic, non-linear, non-parametric and chaotic* [29].The non-linearity of the data, the high volatility and the large number of external factors lead the results obtained through common classifiers to be close to random[30], making this task very challenging to address. Moreover, when employing Frameworks (e.g., *Keras*[1]) and hardware architectures (e.g., *CUDA*[2]) to speed up computation through GPUs, results may change at each run and cannot be easily reproduced [31].

Essentially, Machine Learning approaches cover most of the research achieved in the field but, despite the numerous and sophisticated techniques available today, such a task continues to be considered challenging [32]. There are several reasons to explain that: (i) existing methods employ classifiers whose intrinsic parameters are tuned without a general approach, but are based on values heavily depending on the used classifier and the target data [15, 33]; (ii) the lack of a general technique to set the hyper-parameters (*e.g.* training and test set sizes, lags and walks dimension) for the experiments usually makes them not reproducible and, thus, difficult to assess and to compare with benchmarks or other approaches [33]; (iii) several works in literature do not specify whether they are performing their test analysis on in-sample or out-of-sample data, and this is a further reason of confusion [34, 35, 15]; (iv) several works employ classifiers without stating clearly which is the best and under which conditions [1]. This may bring to the common sense that each proposed classifier exploits the peculiarities of the presented market data. Therefore, this does not help understanding whether the method is effective or there are *ad-hoc* classifiers and data choices to report best results only [26, 36]; (v) different combinations of feature selection techniques have been explored in Financial Forecasting, but a Framework

[1]https://keras.io/

[2]https://developer.nvidia.com/cuda

that can be designed with the goal to get as input any feature and generate the optimal number of output features is still missing [37]; (vi) for the evaluation step, several metrics have been proposed but there have not been precise explanations on the adoption of one with respect to the other. This further introduces confusion on the overall analysis and on which metric should be prioritized [38, 39, 40, 41]; and (vii) to define trading strategies, parameters such as those for the data preparation, algorithm definition, training methodology and forecasting evaluation must be the choices to be made by the trading systems architect [6]. To the best of our knowledge, the literature does not offer Financial Market forecasting approaches based on a systematic strategy, able to model automatically itself with regard to these parameters and chosen market in order to perform well the forecasting task no matter the market considered.

In this thesis, the use of Machine Learning and Deep Learning has been applied in the Finance domains through not yet experimented methodologies. Precisely, starting from some of state-of-the-art tools, in an early phase of my PhD a general approach for Financial Forecasting with application to technical analysis, based on an ensemble of predictors automatically created, considering any kind of classifiers and adjustable to any kind of market, has been studied and proposed. Secondly, the use of more sophisticated tools like Deep Convolutional Neural Network and Double Q-Learning (DQL) agents and Deep Q Networks (DQN) as been applied to perform classification task to the Financial Markets in order to forecast whenever the market goes up or down. The thesis contributes to the scientific literature in terms of results and resources. Furthermore, the work of this thesis has been carried out within the company Visioscientiae SRL. The proposed algorithms are currently being tested for a production release.

In brief, the research program addressed during the PhD course was dedicated to the study and development of methodologies to forecast the Financial Markets methodologically and in a scientific way.

The main research questions addressed in the target domains are:

Q1. How to use existing AI tools and technologies to retrieve useful information in order to forecast the Financial Markets?

Q2. Is there a way to perform automatic tuning of all parameters of an AI model that is generalizable enough to be applicable to any Machine Learning model and Financial Markets?

Q3. There is a scientific and reproducible way to reduce the risk of an investment made through AI approach?

Q4. How different techniques and their combinations will impact the performances of the final predictor? Does using ensemble strategies systematically improve performance?

By answering to these research questions, the contributions provided in this research work are:

- A novel Auto-Configurable ensemble algorithm to significantly improve the performance of ML approach applied to Financial Forecasting;
- A novel approach for the forecasting of market behaviour by using Deep Learning technologies and by encoding time series to GAF images. The developed CNNs have been applied to the GAF images for a classification task.
- A step forward in efficient stock trading with ensembles by presenting an approach that uses two well known and efficient Machine Learning approaches, namely Deep Learning and Deep Reinforcement Learning, in a three layer fashion.
- A tool called *Hawkeye* which provides a Visual Framework for in-depth analysis of results obtained from Deep Learning approaches, tackling classification tasks within the financial domain and aiming at a better interpretation and explanation of the trained Deep Learning models.

1.2 Dissertation Structure

The remainder of this thesis is organised as follows:

- Chapter 2 introduces backgrounds and basic concepts about Machine Learning and Deep Learning and other technologies that have been adopted across the various addressed research problems. Moreover, it describes which software tools and libraries have been adopted. Finally, the used metrics will be explained;
- Chapter 3 analyzed the state of the arts about Machine Learning and Deep Learning applied to Financial Forecasting;
- Chapter 4 proposes a fully automated optimized ensemble approach, where an optimized feature selection process has been combined with an automatic ensemble Machine Learning strategy, created by a set of classifiers with intrinsic and hyper-parameters learned in each marked under consideration. In other words this chapter introduces a general approach for Financial Forecasting with application to technical analysis based on an ensemble of predictors automatically created, considering any kind of classifiers and adjustable to any kind of market. This work has been published on the [1] **Computation** Journal;

- Chapter 5 presents a novel approach which aims to achieve market prediction over the SP500 index, by using an ensemble of Convolutional Neural Networks, with the training phase executed over Gramian Angular Fields images (particularly, the GADF). Comparisons both with state of the art benchmark approaches (e.g. Buy & Hold strategy) and with the results of an existing competitor method have been performed, showing that the proposed approach is capable of obtaining a higher profit in the same investment period. This work led to the publication of a paper on the [2] **IEEE/CAA Journal of Automatica Sinica**;

- Chapter 6 proposes an extension of the work proposed in the previous chapter. In particular, this novel approach proposes a Machine Learning based stock trading system composed by three different layers, that leverages both on Deep Learning and Deep Reinforcement Learning benefits. In summary, the first layer is made by stacking trading signals with Convolutional Neural Networks. The output of this first layer is used as input for the Reinforcement Meta Learner second layer. Last layer, propose the fusion of several multiple DQN agents trading signals, in order to take more decisions into account for trading. This work led to the publication of a paper on the [1] **Applied Intelligence** Journal;

- Chapter 7 concludes the novel proposals of this thesis by presenting a Visual Framework to perform an in-depth analysis of the results generated by Deep Learning classifiers. This Visual Framework goes into the direction of the recent *Interpretable AI* trend and it allows for an in-depth analysis of the results obtained from Deep Learning classifiers specialized in the Financial domain. Notably, this Framework also offers advanced metrics for measuring both economic performance and quality of the classification models adopted, along with targeted tools for exploring the results and generating comprehensive reports. This work led to the publication of a paper on the [3] **ACM International Conference Proceeding Series**.

- Chapter 8 closes the circle of the work presented in this Dissertation. This chapter contains conclusions and final remarks regarding the presented approaches within the thesis. Finally, future works are pointed out.

Chapter 2

Technical background

This chapter provides basic concepts regarding Artificial Intelligence. Starting with the history of artificial intelligence, basic concepts about Machine Learning and Deep Learning will be introduced. To better understand the algorithms proposed in this dissertation the most famous Supervised and Unsupervised Machine Learning algorithms will be discussed, followed by a deep overview of Deep Learning. Afterwards, the experiment pipeline in Machine Learning will be described, starting with problem definition and ending with model evaluation. Finally, the evaluation metrics used will be provided and explained.

2.1 History of Artificial Intelligence

The idea of inanimate objects coming to life as intelligent beings has been around for a long time. The ancient Greeks had myths about robots, and Chinese and Egyptian engineers built automatons. The beginnings of modern AI can be traced to classical philosophers' attempts to describe human thinking as a symbolic system. But the field of AI wasn not formally founded until 1956, at a conference at Dartmouth College, in Hanover, New Hampshire, where the term *Artificial Intelligence* was coined. Today, due to the rise of Big Data and improvements in computing power, Artificial Intelligence has entered the business environment and public conversation [42].

Originally, the early pioneers of AI believed that every aspect of learning or any other feature of Artificial Intelligence can in principle be so precisely described that a machine can be made to simulate it. Therefore, symbolic AI took center stage and became the focus of research projects. Scientists developed tools to define and manipulate symbols. Symbolic AI programs are based on creating explicit structures and behavior rules and the data must be processed according to these rules. Symbolic Artificial Intelligence showed early progress at the dawn of AI and computing. Symbolic Artificial Intelligence is very convenient for settings where the rules are very clear cut; in fact, rule-based systems still account for most computer programs today, including those used to create Deep Learning applications. However, symbolic AI starts to break when you must deal with the messiness of the world and came up to be inflexible to face more intelligent tasks, such as object recognition, content categorization, and machine translation [43].

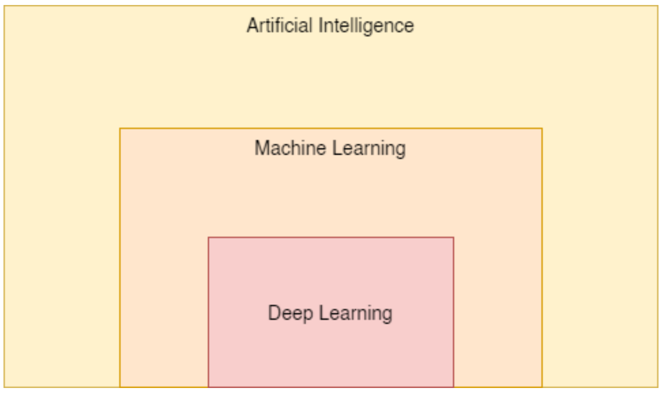


Figure 2.1: Example of hierarchy in Artificial Intelligence field

To address these issues it was investigated how machine can learn patterns on its own, starting from data relevant for the targeted task. So the new AI paradigm was born, called Machine Learning. ML teaches computers to think in a similar way to how humans do: learning and improving upon past experiences. It works by exploring data and identifying patterns and involves minimal human intervention. In other words, humans provide the computer with data and labels data and the machine learns these patterns; then, the computer can recognize these patterns even on data it has never seen before.

Machine Learning uses three main techniques:

- **Supervised Learning**, where data and label are provided by the human and the computer tries to identify patterns and correlations between them;
- **Unsupervised Learning** where only unlabeled data are provided by the human and the computer tries to learn some inherent structure from that data;
- **Reinforcement Learning** where an agent receives information about its environment and learns to choose actions that will maximize some reward.

Over time, more sophisticated and complex approaches have been developed. Deep Learning is a class of Machine Learning algorithms that uses Neural Networks to discover patterns from data and learn correlations from them. Learning process is done through multiple and stacked layers, unlike traditional Machine Learning, which learns only one or two layers of data representations. Hence, learning means optimizing the weights of all layers, such that the network correctly maps inputs to expected targets. Given the predicted and true targets, the system computes a score through a *loss function* which measures how well the network is classifying samples. The score is then used by the optimizer that, through a Back-propagation algorithm, arranges the weight values, so that the loss score will be lower in the next iteration. Repeating the loop a sufficient number of times makes it possible to learn weight values that minimize the loss, obtaining a trained model [43].

2.2 Supervised approaches

Learning from past experiences is an attribute of humans while computers do not have this ability [44]. In supervised Machine Learning, our main goal is to learn a target function that can be used to predict the class of a sample or group of samples. In other words, Machine Learning is a type of Artificial Intelligence technique that makes the system automatically obtain knowledge from data with no explicit programming. Using supervised approaches, human involvement focuses only on specifying how the computer accesses the data allowing it to learn by itself. The main goal is to enable systems to learn automatically with no human intervention [45]. The process of applying supervised ML to a real-world problem is described in the figure below.

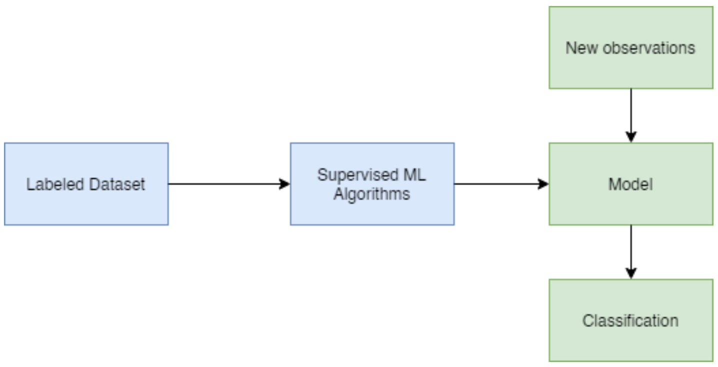


Figure 2.2: Machine Learning pipeline process.

Supervised Machine Learning approaches use information learned from data (*i.e* training set data) to classify future samples. In supervised learning the first step is dealing with the dataset. In order to perform better training, a good feature selection of the dataset could be necessary. Data preparation and data preprocessing is a key point in supervised Machine Learning. During the training process, ML develops an inferred function to forecast the output values. The system is capable of providing results to input data with an adequate training process. ML algorithm compares the obtained results with the actual and expected results to identify errors to change the model based on results. Though several types of ML techniques are available, supervised ML approaches are the most popular and commonly used technique.

Some of the most famous supervised Machine Learning algorithms are described below:

- **Support Vector Machine.** Support Vector Machine (SVM) is a very well know algorithm mostly used to solve a binary classification problem. It works by defining a boundary through hyperplane in order to separate a class from the other maximizing the margin between the two classes. This hyperplane is used to classify new sets of data. SVM usually works with a 2-dimensional plane but it can be used as well with a multidimensional hyperplane. The algorithm takes labeled pairs (x_i, y_i) where x_i is a vector representation of input data, and y_i is a numerical label. The algorithm then applies an optimization function in order to separate classes.

- **Support Vector Machine + Stochastic Gradient Descent (SVM+SGD).** This method extends the standard SVM implementation including the *Stochastic Gradient Descent* algorithm during training. SGD finds the best coefficients describing the decision boundaries through a classification function which minimizes a hinge loss function and allows performing training over large data while reducing the computation time [46].

- **Decision Trees.** Decision Trees (DT) is a type of supervised Machine Learning used to categorize or make predictions based on how a previous set of questions were answered. DTs involve representing a set of classification rules through a tree: a hierarchical structure consisting of a set of nodes, connected by oriented and labeled arcs. The root node and the inner nodes represent attributes, the leaf nodes represent class labels.

  - **Root node**: no incoming node, two or more outgoing branches labeled with the possible values of the attribute values
  - **Inside nodes**: one incoming branch, two or more outgoing branches labeled with the possible values of the attribute
  - **Leafs**: one incoming branch, no outgoing branches.

  Each root-leaf path represents a classification rule and several trees can describe the same dataset. There are heuristics based on local optimization criteria that make it possible to induce a sufficiently small and accurate tree. The basic idea is to choose as a root the attribute that allows the classes to be more discriminating, the subtrees are constructed by applying this criterion recursively.

- **Random Forests.** Random Forests (RF) method is a popular Machine Learning algorithm highly applicable to various classification problems. RF consists of many decision trees, where each tree is independently trained and votes for a class for the data presented as an input [47]. Essentially, each decision tree splits data into smaller data groups based on the features of the data until there are small enough sets of data that only have data points with the same label. These splits are chosen according to a purity measure and, for each node, the algorithm tries to maximize the gain computed on it. The final decision is made by an ensemble (majority vote) of the trees.

- **Naive Bayes.** Naive Bayes (NB) classifiers are widely used to perform classification tasks. Naive Bayesian Networks (NBN) are very simple Bayesian networks that are composed of directed acyclic graphs with only one parent (representing the unobserved node) and several children (corresponding to observed nodes) with a strong assumption of independence among child nodes in the context of their parent An advantage of the Naive Bayes classifier is that it requires a small amount of training data to estimate the parameters necessary for classification.

- **Neural Networks.** Neural networks attempt to simulate (in a very simplified way) how the human brain learns. In brief, a typical Neural Network has from a few dozen to thousands or even millions of artificial neurons arranged in a series of layers, each of which connects to the layers on either side. The first layer also known as *input layer* takes the data as input for processing and the last layer called *output layer* returns the predicted value of the inputs; all other levels between these two are grouped under the name of *hidden layers*. The connections between these nodes are weighted, and the weight value is calculated and modified during the training phase by the back-propagation algorithm. More details about modern Neural Networks methods have been deeply described in Section 2.4.

- **Grandient Boosting.** Gradient Boosting is one of the most powerful and popular techniques for building predictive models. With GB a model is created in a gradual, additive, and sequential way, and it is generalized by allowing the optimization of an arbitrary differentiable loss function [48]. Precisely, GB involves three elements: a loss function to be optimized, a weak learner to make predictions and an additive model to add weak learners to minimize the loss function.

2.3 Unsupervised approaches

Supervised algorithms assume that the data contains the Y label for each sample in order to build the model. In many cases, however, it is impossible to have a labeled dataset because labeling may be expensive, error-prone, or sometimes simply not possible. Unsupervised Machine Learning is used in these cases because it's a type of ML that does not aim to classify samples but instead tries to group samples according to a similarity criterion. It can be applied to any kind of data because it does not need a training stage. One of the most common unsupervised approach is named *clustering*, which is aimed to segment a collection of samples $X = \{x_1, ..., x_n\}$ into partitions $C = \{c_1, ..., c_m\}$ called *clusters* such that each point in a cluster is similar to points from its own cluster than with points from some other cluster.

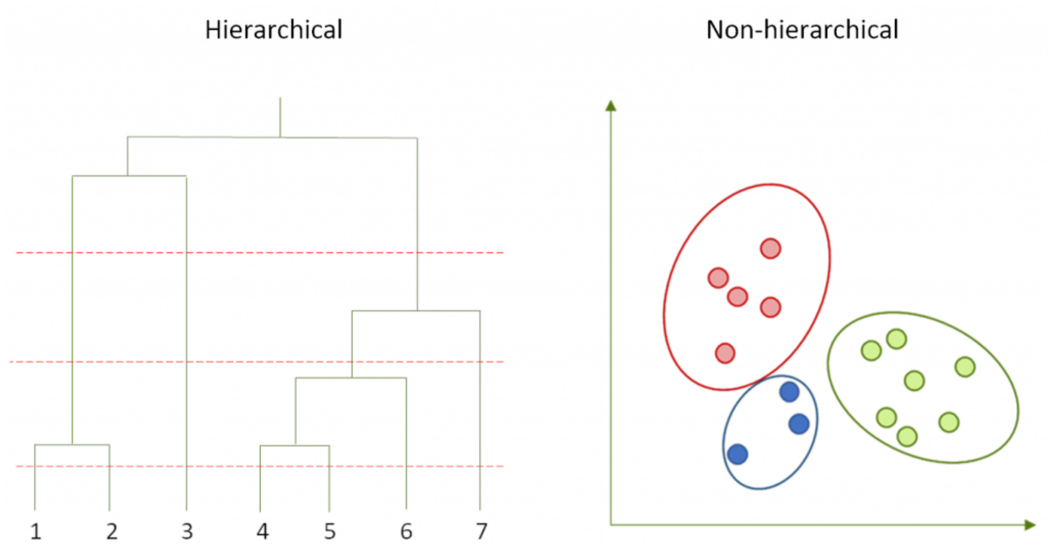


Figure 2.3: Example of clustering algorithms

Some of the most famous unsupervised Machine Learning algorithms are described below:

- **K-means Clustering.** The K-means Clustering is a partition method that aims to group similar data points, discovering hidden patterns dividing the dataset into a number of K clusters. The algorithm works by randomly initializing a defined number of clusters that is provided as an input parameter and iterating the centroid optimization process several times according to similarity criteria like the sum of square distance. The centroids are like the heart of the cluster, the algorithm finds the closest points and adds them to the cluster. The output of the algorithm would be a set of labels assigning each sample to one of the K groups.

- **Hierarchical Clustering.** Hierarchical clustering is similar to regular clustering, except that its aim is to build a hierarchy of clusters that means, in other words, a tree of clusters which is usually called *dendrogram*. This can be very useful if one wants flexibility in how many clusters one ultimately wants. Each cluster contains children that are clusters as well, except for the leaves of the tree. A hierarchical clustering algorithm can be either *agglomerative* or *divisive*. In terms of outputs from the algorithm, in addition to cluster assignments, it also builds a nice tree that gives information about the hierarchies between the clusters. In this way, it is possible to pick the right number of clusters one needs.

- **Principal Component Analysis** The principal component analysis (PCA) is a dimensionality reduction method that is often used to reduce the dimensionality of large datasets. The high dimensionality of the data can dramatically impact the performance of ML algorithms and also a large number of dimensions in the feature space can mean that the volume of that space is very large, and the points that are in that space often represent a small and non-representative sample. PCA brings the dimensionality of the data up to two dimensions and this is done through a linear transformation of the variables that projects the original ones into a new cartesian system. The reduction in complexity is achieved by simply analyzing the main ones, by variance, among the new variables. These basis vectors are called *principal components* (PCs), and the selected subset constitutes a new space that is smaller in dimensionality than the original space but maintains as much of the complexity and information of the original data as possible.

2.4 Deep Learning

One of the limitations of Machine Learning is that as the amount of data increases, the efficiency of the algorithms' predictive models decreases. Deep Learning emerged as a subclass of a broader family of Machine Learning, where algorithms inspired by the structure and function of the brain, called artificial Neural Networks, are applied for pattern classification and regression tasks. The usage of Deep Learning and in particular of Neural Networks has grown more and more in recent years, due to several factors including the increase of computational abilities of computers through the use of Graphical Processing Units (GPUs) [49], and many services we use on a daily basis actually rely on these technologies. DL eliminates some of the data pre-processing that is typically involved with Machine Learning. These algorithms can process an enormous amount of unstructured data, like text and images, and it automates feature extraction, removing some of the dependency on human experts. In brief, a DNN consists of multiple numbers of layers of interconnected nodes (called *perceptrons*, each building upon the previous layer to refine and optimize the prediction or categorization. In this paragraph, we will introduce the most famous types of Neural Networks present in the literature, some of which have been adopted in this dissertation.

A **Feed-forward Neural Networks** (FNN) is an artificial Neural Network where connections between units do not form loops, unlike recurrent Neural Networks. This type of Neural Network was the first and simplest of those developed. In this Neural Network the information moves only in one direction, forward from the input nodes, through hidden nodes (if existing) to the output nodes. There are no cycles in the network. The feed-forward networks do not have the memory of inputs that occurred at previous times, so the output is determined only by the current input. The nodes of this NN are called *perceptrons* and they are randomly joined by weighted connections in a many-to-many fashion. On the basis of the input values fed into the network, nodes of a certain level can be activated and their signal is broadcasted to the subsequent level. In order to activate nodes of a subsequent level, the signal generated at a given level is weighted and must be greater than a given threshold. Weights are generally initialized with random values and adjusted during training in order to minimize a predefined objective function. A simple schema of a three-layer Feed-forward Neural Network model is shown in Figure 2.6.

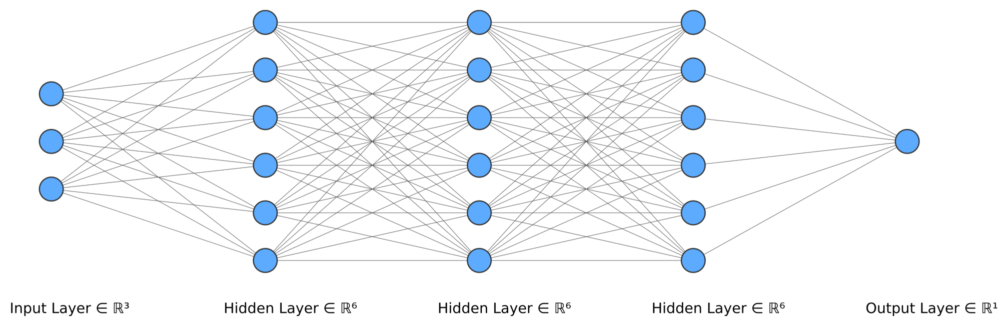


Figure 2.4: An example of a Feed-forward Network composed by three hidden layers.

The sample FNN in 2.6 accepts three-dimensional inputs and returns one-dimensional outputs. Each node of a given layer is connected to nodes of the subsequent layer. The input data is fed into the network by means of Layer 1, which acts as Input Layer, and then sent to the first hidden layer, i.e., Layer 2. The output of this layer is finally propagated to Layer 3, which represents the Output Layer. The action to move data from a layer to another by activating or not the corresponding nodes is generally called *forward pass* of the network.

Convolutional Neural Networks (CNNs) is a type of feed-forward artificial Neural Network whose neurons functioning is inspired by the way the animal visual cortex works. Convolutional Neural Networks are distinguished from other Neural Networks by their superior performance with image, speech, or audio signal inputs. They have three main types of layers, which are: convolutional layer, pooling layer and fully-connected layer. The convolutional layer is the most relevant block of a CNN, and it is where the majority of computation occurs. It requires a few components, which are input data, a filter, and a feature map. A convolution is a linear operation that involves the multiplication of a set of weights with the input; the filter is smaller than the input data and the type of multiplication applied between a filter-sized patch of the input and the filter is a dot product. The convolutional layer is always the first layer of a convolutional network and the fully-connected layer is the final layer. Regarding convolutional layers, they can be followed by additional convolutional layers or pooling layers. More details on how the convolutional layer works will be given later.

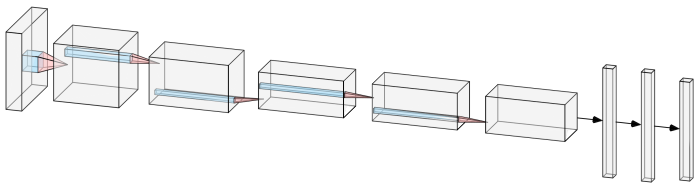


Figure 2.5: An example of a Convolutional Neural Network.

Recurrent Neural Networks (RNN) is a class of ANN where neurons are connected together in a loop. In contrast to what we have discussed with the FNNs which commonly pass the input data directly from input to output nodes, RNNs have cyclic or recurrent connections among nodes of distinct levels. This interconnection among layers allows the use of one of the layers as a memory of state, and allows, providing a time sequence of values as input, to model a temporal dynamic behavior dependent on the information received at previous times. RNNs are applicable to tasks of predictive analysis on sequences of data, such as the recognition of the handwriting or the vocal recognition. With RNNs, it is possible to use inputs of arbitrary length, overcoming the limitations of other Neural Networks, such as convolutional Neural Networks which impose fixed-length inputs.

A very well-known RNN is **Long Short-Term Memory** (LSTM) network. LSTM networks are designed to work on time series and have achieved state-of-the-art results on challenging prediction tasks. LSTM networks have cyclic connections among nodes like RNNs with the addiction of memory blocks in their recurrent hidden layers. Memory blocks save the current temporal state during training and make it possible to learn temporal observations hidden in the input data. The use of an LSTM network is very effective when one wants to solve tasks in which the sequencing and temporal order of the samples are important. For this reason, LSTM networks have had a positive impact on sequence prediction tasks like speech recognition, connected handwriting recognition, financial time series prediction and anomaly detection in network traffic or IDSs (intrusion detection systems).

We will now discuss the various and most important layers present within the hidden layers of the Neural Networks.

Convolutional Layer (CL)

Convolutional networks take their name from the convolutional layer, which is the core of the architecture of this type of Neural Network. The convolution operation is a linear operation involving the multiplication of a set of weights called *filters* or *kernels* or *feature detector* to the input data (usually images). These kernels are intentionally smaller than that of input data as it allows the same set of filters to be multiplied by the input array multiple times at different points on the image. In simple words, the filter is applied systematically to each filter-sized input data from left to right and top to bottom. It is important to note that filters act as feature detectors from the original input image. Convolution preserves the spatial relationship between pixels by learning image features using small squares of input data.

Normalization Layer (NL)

Internal covariate shift (ICS) is the technical name used in Machine Learning to refer to the change in the distribution of inputs into the various layers of the Neural Network. Training Deep Neural Networks with tens of layers is challenging due to several factors and one of them is initial random weights and the distribution of the inputs to deeper layers in the network may change after each mini-batch when the weights are updated. *Batch Normalization* (BN) is a common technique for training very Deep Neural Networks and solve the ICS problem. It reduces the number of training epochs required to train Deep Neural Networks standardizing the inputs to a layer for each mini-batch. Standardizing the inputs mean that inputs to any layer in the Neural Network should have approximately zero mean and unit variance. BN layer normalize each input in the current *mini-batch* by subtracting the input mean in the current mini-batch and dividing it by the standard deviation.

Other Layers

There are also other layers that can be leveraged in order to fine-tune the performance of a model. The most representative ones are described below.

Pooling layer Pooling layers are similar to convolutional layers, but they perform a specific function in order to reduce the dimensionality of the network. Max pooling takes the maximum value in a certain filter region and average pooling takes the average value in a filter region.

Noise Layer. A Noise layer is usually employed to avoid model over-fitting. It consists in modifying a fraction of the input of layers, adding and subtracting some values following a predefined distribution (e.g., Gaussian).

Dropout Layer. A Dropout layer may be seen as a particular type of noise layer. These layers randomly switch off certain neurons in the network. The probability of a neuron being switched off is given by a specific parameter. The dropout layer is often used to include randomness in the training phase and avoid overfitting of the model.

Dense - Fully Connected Layer (FLC). At the bottom of a Neural Network there are Fully Connected layers. In Neural Networks, FCL layers are those layers where all the inputs from one layer are connected to every activation unit of the next layer. In most popular Machine Learning models, the last few layers are full connected layers that compile the data extracted by previous layers to form the final output.

2.5 Experimental Workflow

Dealing with Machine Learning problems requires several steps, some of them optional, to reach the solution. In this section, the different elements of the pipeline will be analyzed, starting from the formulation of the problem to the analysis of the obtained results.

Problem Definition

Starting from the very beginning, the basic idea behind Machine Learning is that a given set of data contains enough information to be used to predict an output. A Machine Learning model must therefore find the relationships and patterns hidden within the data. Therefore, the first step must be defined: what kind of problem do you want to solve? Binary classification, multi-class, regression, etc. This first decision will then lead to the definition of the next steps such as the model architecture, the loss function and so on. In addition, inputs and outputs need to be defined and, based on that design choice, proper training data should be retrieved. [43].

Data Pre-Processing

Data pre-processing is a crucial step in obtaining good results with ML. This macro category includes operations such as *vectorization*, *normalization*, missing values handling, *feature extraction* etc.

- *Vectorization.* ML algorithms take vectors of numbers as input feature. Therefore, it is important to turning data into a numerical data matrix, where each row is associated with a sample, and the columns are the feature of that sample. This process is called vectorization. Feature like words or text sentence can be represented as a list of integers. On the other hand, this step is not needed when data is already in numerical forms, like images, which are numerical matrices.

- *Normalization.* Data Normalization is a common technique applied as part of data preprocessing for Machine Learning. The goal of normalization is to change the values of numeric columns in the dataset to use a common scale, without distorting differences in the ranges of values or losing information. Normalization is also required for some algorithms to model the data correctly. Depending on the task and the inputs, the data should have values in [0,1] range. Image data encoded as integers in the range [0,255] is usually cast to float and divided by 255 so that they become float values in the [0,1] range. Similarly, when predicting users' identities, each feature could be normalized to have a standard deviation of 1 and a mean of 0.

- *Missing Values Handling.* In any real-world scenario, there are always few missing values within the dataset and no model can handle these missing values on its own. Missing values can be imputed, which means we can use information in the training set predictors. There are many options when we decide to replace a missing value. A good practice can be to replace missing values with a constant value 0 or use a value from another randomly selected sample. To estimate the missing value can be used mean, median or mode value otherwise can be used a value from by another predictive model.

Build the Model

After defining the problem and pre-processing the data, the next step is to develop a model capable of solving the problem. Three key choices to build the model should be considered: (i) model architecture that should learn meaningful data representations, (ii) differentiable optimization function that should match the type of problem, (iii) optimization configuration that should support the model in minimizing the objective function [43]. In Deep Learning, model optimization takes place through a repetitive process called *epoch*, which calculates the current error e_i of the model at the epoch E_i. The function that calculates the error is called *loss function.* Such a function is used to estimate the loss of the model so that the weights can be updated to reduce the loss on the next evaluation. There are several loss functions to choose for each ML problem. For instance, for regression problems, common loss functions are *Mean Squared Error*, *Mean Squared Logarithmic Error*, and *Mean Absolute Error*; for binary supervised classification, *Binary Cross-Entropy*, *Hinge Loss*, and *Squared Hinge Loss* are usually adopted; for multi-class classification, common solutions are *Multi-Class Cross-Entropy*, *Sparse Multi-Class Cross-Entropy*, and *Kullback Leibler Divergence.* Finally, within DL, an optimizer is integrated to update the weights and minimize the loss function. The loss function is used by the optimizer to move in the right direction to reach the global minimum. Common optimizers include *Root Mean Square Propagation* (RMSProp) and *Adaptive Moment Estimation* (ADAM).

Model Evaluation

The standard practice for building and evaluating a model is to not use the entire dataset for the training set. In order to have a model that is capable of generalization and not subject to overfitting, it is important to split the data into (i) a training set, (ii) a validation set, and a (iii) test set. The training set contains a big subset of the data to build the predictive models. Validation set is a portion of the data used to assess the performance of the model built in the previous phase. It provides a test platform for fine-tuning a model's parameters and selecting the best-performing model. Test set contains only unseen data and it is used to evaluate the model. The test set should be used only when the tuning of the parameters has been completed, to ensure the smallest chance of overfitting.

- *Held-Out Validation.* The dataset is split into two parts, one part for the training set (80%-90%) and one part for the test set (20%-10%). The model is built on the training set, and its performance is evaluated on the test set. This validation technique is quite simple, and a sufficiently large number of samples is required to be statistically representative, preventing the validity of the experimental results.

- *k-Fold Validation.* Similar to held-out validation but in this case the data is split into K equal-sized partitions. Then, the model is trained on K–1 partitions and evaluated on partition i. The process is repeated K times, with a different partition i as a test set. The final metrics are averaged to obtain the final score. This might solve issues related to significant variance on final metrics over different train-test split

- *Walk-Forward Validation.* Since the chronological order of data is important in time series, a walk-forward optimization strategy is used in Financial Forecasting. WFO splits historical data into the following two types: (i) *In-Sample Data* (IS) used for initial testing and parameters optimization and *Out-of-Sample Data* (OOS) reserved for data set which is not a part of the *in-sample* data. IS is divided into multiple chunks called walks where each one is divided into two parts: *Training* and *Testing* sets. The optimization process is then repeated for each walk and then the best parameter combination is tested on OOS. There are two types of WFO strategy: (i) *Non-anchored Walk Forward Optimization* and (ii) *Anchored Walk Forward Optimization.*

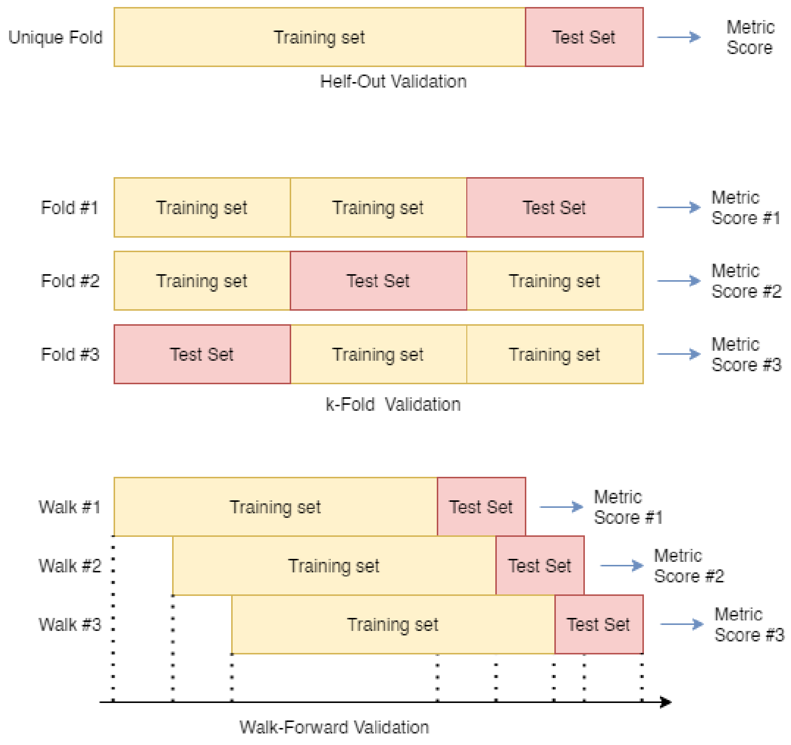


Figure 2.6: ML Evaluation techniques: held-out (top), k-fold (middle) walk-forward (bottom)

2.6 Evaluation Metrics

This section will explain the evaluation metrics used to assess the performance of the novel approaches proposed within this thesis. In Chapter 2.6.1 will explain evaluation metrics for Machine Learning models while Chapter 2.6.2 will introduce evaluation metrics specific for the financial domain.

2.6.1 AI Metrics

Accuracy

The *Accuracy* gives us information about the number of instances correctly classified, compared to the total number of them. It provides an overview of the classification performance. Accuracy is a well-known metric in Machine Learning; it is defined as the ratio between the number of correct predictions against the total number of test samples.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.1}$$

where:

- **TP** are correctly labelled positive class samples;
- **TN** are correctly labelled negative class samples;
- **FP** are the samples belonging to the negative class labelled as positive;
- **FN** are the samples belonging to the positive class labelled as negative.

Precision

Precision is a measure that is often used for evaluating supervised methods. It indicates how many items have been correctly classified for a given class considering all elements that have been classified with that class. It is expressed by the following equation:

$$Precision = \frac{TP}{TP + FP} \tag{2.2}$$

In equation (2.2) TP is the number of items correctly classified for the target class and FP is the number of elements that have been erroneously classified for that class.

Recall

Recall is a metric that is often computed together with Precision. It is defined by the equation (2.3), and measures how many items that belong to a given class have been properly classified for the class.

$$R = \frac{TP}{TP + FN} \tag{2.3}$$

As for the Precision, in equation (2.3) TP is the number of items correctly classified for the target class. FN is the number of elements that belong to the target class but have not been labeled for that class.

F-measure

A measure often used to combine Precision and Recall is the F-measure. It is computed as their harmonic mean as shown in equation (2.4).

$$F = 2 \cdot \frac{P \cdot R}{P + R} \tag{2.4}$$

2.6.2 Financial Metrics

Return

The *Return* or *Portfolio Return* or *Net Profit* is the effective gain or loss realized by an investment. The return is expressed in USD.

Maximum Drawdown

The *Maximum DrawDown* (*MDD*) represents the largest drop from a peak to a trough before a new peak is reached. It indicates the downside risk in the time period taken into account [50]. Formally, denoting as P the peak value before the largest drop, and as L the lowest value before a new high established, its formal notation can be simplified as shown in Equation 2.5.

$$MDD = \frac{(P - L)}{P} \tag{2.5}$$

Return Over Maximum Drawdown

The *Return Over Maximum Drawdown* (*Romad*) is a metric largely used within the financial field in order to evaluate the gain or loss in a certain period of time, such as the *Sharpe Ratio* or the *Sortino Ratio* [51, 52]. More formally, it represents the average return for a portfolio expressed as a proportion of the *Maximum DrawDown* level, as shown in Equation 2.10, where *Portfolio Return* denotes the difference between the final capital and the initial one.

$$Romad = \frac{Portfolio\ Return}{MDD} \tag{2.6}$$

Equity Curve

The *Equity Curve* (*EC*) reports the change in the value of a trading account in a time period graphically [53]. A significant positive slope usually indicates that the effectiveness of the adopted trading strategy, while a negative slope indicates that such a strategy generates negative returns. For instance, given an *Initial Investment* (*II*) to trade a *number of futures* that have a certain *entry price* and *exit price*, considering the related *trade commission*, is it possible to can calculate the points $EC = \{ec_1, ec_2, \cdots, ec_N\}$ that compose the equity line. An example of *Equity Curve* can be seen shown in Equation 2.7.

$$\begin{aligned}
&ec_1 = II - ((entry\ price\ \times\ number\ of\ futures) - commission)\\
&ec_2 = II - ((exit\ price \times\ number\ of\ futures) - commission)\\
&\vdots\\
&ec_{N-1} = II - ((entry\ price\ \times\ number\ of\ futures) - commission)\\
&ec_N = II - ((exit\ price \times\ number\ of\ futures) - commission)
\end{aligned} \tag{2.7}$$

Coverage

The *Coverage* metric reports how many times in percentage an operation is performed [54] (i.e., *buy* or *sell*) on the market (i.e., $|buy| + |sell|$), compared to the number of days taken into consideration, as shown in Equation 2.8. It gives us important information since in addition to predicting a *buy* or *sell* operation, an algorithm can also predict to not buy and not sell anything ($hold$).

$$Coverage = \frac{(TP + FP)}{TP + TN + FP + FN} \tag{2.8}$$

Sharpe Ratio

The *Sharpe Ratio* (also known as the *Sharpe index*, the *Sharpe measure*, and the *reward-to-variability ratio*)is a financial risk index used to help investors to understand the return of an investment compared to its risk; the greater the value of the Sharpe Ratio, the more attractive the risk-adjusted return;

$$SharpeRatio = \frac{R_p - R_f}{\sigma_p} \tag{2.9}$$

Where:

- R_p is the actual or expected portfolio return;
- R_f is the Risk-free rate;
- σ_p is the standard deviation of the portoflio's excess return

Sortino Ratio

The *Sortino Ratio* is a variation of the *Sharpe Ratio*2.6.2 that differentiates harmful volatility from total overall volatility by using the asset's standard deviation of negative portfolio returns—downside deviation—instead of the total standard deviation of portfolio returns. The SR takes an asset or portfolio's return and subtracts the risk-free rate, and then divides that amount by the asset's downside deviation.

$$SortinoRatio = \frac{R_p - R_f}{\sigma_p^D} \tag{2.10}$$

Where:

- R_p is the actual or expected portfolio return;
- R_f is the Risk-free rate;
- σ_p^D is the standard deviation of the downside

2.7 Software Tools and Technologies

During the various research works the following toolkits have been employed for the development and evaluation of the proposed solutions.

Scikit-learn[1] is a Python library that provides implementations of supervised and unsupervised algorithms, and metrics to evaluate results.

Tensorflow[2] is an end-to-end open-source platform for Machine Learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML-powered applications.

Keras[3] is a high-level Neural Networks API, written in Python. It was developed to rapidly build Deep Learning architectures by pre-developed modules. It also has the advantage that can be run on both CPUs and GPUs.

Numpy[4] is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

[1] `https://scikit-learn.org/stable/`
[2] `https://www.tensorflow.org/`
[3] `https://keras.io/`
[4] `https://numpy.org`

Pandas[5] is a Python library that provides easy-to-use data structures and data analysis tools. It allows to efficiently manage great matrices of data and perform classic operations that are typically adopted on tables (e.g., *join* and *selection* of rows based on their values).

Matplotlib[6] is one of the most popular Python library. It is a comprehensive library for creating static, animated, and interactive visualizations in Python.

OpenCV[7] is a huge open-source library for computer vision, Machine Learning, and image processing. OpenCV supports a wide variety of programming languages like Python, C++, Java, etc. It can process images and videos to identify objects, faces, or even the handwriting of a human.

[5] https://pandas.pydata.org/
[6] https://matplotlib.org/
[7] https://opencv.org/

Chapter 3

State of the art

The Neural Networks approaches applied to Financial Forecasting started approximately across the end of 80's and the beginning of 90's. In those years, one of the first Artificial Neural Network has been proposed [7], with the aim of predicting the TOPIX index (Tokyo Stock Exchange Price Indexes), by taking six metric vectors as input. In the same year, the authors in [55] built a recurrent NN for stock price patterns' recognition, exploiting the *triangle* pattern as a clue to the trend of the future stock prices. In [56], a similar approach has exploited eleven market indicators for building and training a recurrent Neural Network for monthly transition of the stock price index [57]. Several approaches have also been proposed in the field, facing the financial series prediction topic by using the DNM (Dendritic Neuron Model) technique [58], whose performances have been shown both on Asiatic [59] and US Market [60]. These techniques have evolved a lot in the recent 10 years, as well as the finance itself [61]. As a consequence, dozens of scientific papers have been published, proposing approaches which aim at predicting the stock prices by exploiting news data extracted from the most popular social networks for modelling the uncertainty which lies behind the fluctuation of the market [62]. Financial Forecasting is indeed one of the research branches in which Sentiment Analysis found a quite breeding ground [63, 64].

Alongside the Artificial Neural Networks, also the ML approaches had the possibility to show their efficiency through the years[65]. In [66], the authors have compared the capabilities of the Support Vector Machines [67] in market prediction related issues against those obtained by using the Radial Basis Function (RBF) Networks and Back Propagation (BP). In [68], a recent literature review is performed, which compares the modern Machine Learning approaches in FF field.

Interesting results have also been obtained when fusing together the above described techniques: as an example, in [69] the authors have fused ANN with Decision Trees. The rationale behind this hybrid approach is that where ANNs are able to provide quite good performances in forecasting the market trend, a DT model is stronger in generating potential rules which describe the forecasting decisions. Similarly, in [14], a two stage fusion approach is proposed for predicting CNX Nifty and S&P Bombay Stock Exchange Sensex from Indian stock markets. Specifically, the first stage uses a Support Vector Regressor, whereas the second one exploits, in turn, Artificial Neural Networks, Random Forest and Support Vector Regression. Ten indicators have been selected as input to the prediction models. A comparison of different market price prediction approaches is shown in [70], where classification-based approaches usually provide better results, such as [71, 72, 73], rather than some regression-based benchmarks. As reported in [74], moreover, "there is no general consensus on best forecasting technique for price prediction", particularly since "price series is inherently a non-stationary series having non-constant mean and variance", making it not always advantageous to represent the problem through linear models such as regression.

The work in [75] used standardized technical indicators to forecast rise or fall of market prices with AdaBoost algorithm, which is used to optimize the weight of these technical indicators. In [76, 77], the authors used Auto Regressive Integrated Moving Average (ARIMA) in pre-processed time series data in order to predict prices. The authors in [78] proposed a hybrid approach, based on Deep Recurrent Neural Networks and ARIMA in a two-step forecasting technique to predict and smooth the predicted prices. Another hybrid approach is proposed in [79], which uses a sliding-window metaheuristic optimization with the firefly algorithm (MetaFA) and Least Squares Support Vector Regression (LSSVR) to forecast the prices of construction corporate stocks. The MetaFA is chosen to optimize, enhance the efficiency and reduce the computational burden of LSSVR. The work in [80] used Principal Component Analysis to reduce the dimensionality of the data, Discrete Wavelet Transform to reduce noise, and an optimized Extreme Gradient Boosting to trade in Financial Markets. The work in [81] validated an extension of Support Vector Regression, called *Twin Support Vector Regression*, for financial time series forecasting. The work in [82] proposed a novel fuzzy rule transfer mechanism for constructing fuzzy inference Neural Networks to perform two-class classification, such as what happens in Financial Forecasting (*e.g.*, buy or sell). Finally, [83] proposed a novel learning model, called Quantum-inspired Fuzzy Based Neural Network to classification. This learning happens using concepts of Fuzzy c-Means clustering. The reader should notice that fuzzy learning is commonly used to reduce uncertainty in the data [84], so such solutions can be useful for Financial Forecasting. Several other interesting studies have been carried out in the literature, such as a comparison of DL technologies to prices prediction [85], the use of DL and statistical approaches to forecast crisis in the stock market [86], the use of reward-based classifiers such as Deep Reinforcement Learning [87], among others.

However, it is usually known that single classifiers/hybrid approaches can obtain better performance than that of their single versions when applied in an ensemble model [88, 89]. With that in mind, the literature also reports many approaches that exploit a set of different classification algorithms [90, 91, 92] whose results are combined according to a certain criterion (*e.g.*, *full agreement*, *majority voting*, *weighted voting*, among others). An ensemble process can work in two ways: by adopting a *dependent framework* (*i.e.*, in this case, the result of each approach depends on the output of the previous one), or by adopting an *independent framework* (*i.e.*, in this case, the result of each approach is independent) [93]. In this sense, the work in [94] proposed a novel multiscale nonlinear ensemble leaning paradigm, incorporating Empirical Mode Decomposition and Least Square Support Vector Machine with kernel function for price forecasting. The work in [95] fits the same Support Vector Machines classifier multiple times on different sets of training data, increasing its performance to predict new data. Authors of [96] combined results of bivariate empirical mode decomposition, interval Multilayer Perceptrons and interval exponential smoothing method to predict crude oil prices. Other interesting approaches using ensembles are the use of multiple Feed Forward Neural Networks [97], multiple Artificial Neural Networks with model selection [98], among others.

Authors in [99], used modular Neural Networks trained on various technical and economical information in order to determine when to buy and sell stocks. This work was extended later in [100] using recurrent Neural Networks that are more suitable to time series. The work in [101] used a similar recurrent Neural Network, but trained on eleven economic indicators. Other subsequent works have used more advanced ML classifiers to perform stock trading. The work presented in [102] used Long Short Term Memories DNN, an evolution of recurrent networks, on augmented market trading data. The work in [103] used a more evolved recurrent network to predict the stock trading signal by forecasting the opening, closing, and difference between these prices.

Other works have explored the use of Non-Neural Network based classifiers. Authors in [104] used SVM on features originated from the prices, such as trend, market fluctuation, and noise on multiple time resolutions to predict stock trends. Zhou *et al.* [105] employed the same classifier on a very heterogeneous dataset with sources from historical transaction data, technical indicators, stock posts and news to predict the directions of stock price movements. Another kind of classifier usually considered in previous works is the Random Forest, as it consists of an ensemble of individual decision trees and, therefore, can be a powerful tool for trading. The work in [106] evaluated the robustness of RF to stock selection through fundamental/technical and pure momentum feature spaces. Finally, the work of Khan *et al.* [107] has assessed the effectiveness of RF on features from social media and financial news data.

The use of Reinforcement Learning (RL) in stock market prediction has shown state-of-the performance in several works in the literature and is considered a trending topic in stock market prediction. By considering the market as an environment that returns maximized rewards when the right trading signals are emitted, the stock trader agents are trained as much as possible in order to follow the market behavior by optimizing financial metrics. These metrics can be returns, Sharpe Ratio, among others. One of the pioneer works in this regard comes from the work in [108], that used a Q-Learning value-based RL approach to optimize asset allocation decision. Later, Mihatsch and Neunier *et al.* [109] added in the Q-learning function the notion of risk. Gao and Chan*et al.* used as performance functions of Q-learning training the absolute profit and relative risk adjusted profit[110]. Authors in [111] used four Q-learning agents that act serially in different steps of the trading procedure. Moody *et al.* [112] used a Recurrent Neural Network in the RL pipeline for trading, an approach known as Recurrent Reinforcement Learning (RRL). Recent solutions proposed in this aspect are the work in [113], which modified and adapted the A3C RL algorithm and joined it with Deep Learning, and also the work of Lei *et al.*[114], which proposed DL and DRL to adaptively select and reweight several features of financial signals.

Chapter 4

A Holistic Auto-Configurable Ensemble Machine Learning Strategy for Financial Trading

This chapter introduces a general approach for Financial Forecasting with application to technical analysis based on an ensemble of predictors automatically created, considering any kind of classifiers and adjustable to any kind of market. In our ensemble, each market will have two sets of parameters tuned: the time series parameters (hyper-parameters) and classifier parameters (intrinsic-parameters), no matter the classifiers considered in the ensemble. These parameters are tuned in late past (in-sample) and early past (out-of-sample) datasets, respectively. The input data of such an ensemble is transformed by the Independent Component Analysis, whose parameters are also optimized to make it general enough to return the optimal number of output signals. Therefore, this approach is different from the literature as it is composed of an ensemble of classifiers that can include any classifier and can be maximized for more than one market. To do that, this dissertation study the performance of our *data-driven* ensemble construction by considering different performance metrics in known data in order to tune ensemble parameters over the space of features (ICA feature selection), parameters (parameters of classifiers, or intra-parameters), and also in the space of time (parameters of the time series). Experiments performed in several futures markets show the effectiveness of the proposed approach with respect of both buy-and-hold strategy and other literature approaches, highlighting the use of such a technique specially by conservative and beginner investors who aim to do safe investment diversification.

The contributions of this work are therefore the following:

1. We formalise a general ensemble construction method, which can be evolved by considering any kind of different classifiers and can be applied to any market.

2. We propose an auto-configurable nature, or *data-driven* nature of such an ensemble. Our approach seeks for hyper-parameters and intrinsic-parameters in late and early past data respectively, generating a final ensemble no matter the market considered.

3. We discuss the use of an optimized Independent Component Analysis (ICA) method as feature selection of the ensemble input, in order to produce the best number of selected features given any number of input signals.

4. We conduct a performance study by using different metrics based on classification, risk and return, comparing our approach to the well established Buy and Hold methodology and several canonical state-of-the-art solutions.

5. In order to reduce the risk that the general strategy optimization phase would lead to results affected by overfitting bias, this dissertation systematically rely on the concepts of strictly separated in-sample and out-of-sample datasets for an efficient two-step ensemble parameter tuning, aimed to trade in Financial Markets.

4.1 Proposed Approach

This dissertation proposes an auto-configurable ensemble, composed by any number of classifiers and adjustable to any market. This ensemble is created automatically after optimizing two sets of parameters: hyper-parameters and intrinsic-parameters. Once optimized in an In-Sample late past (`IS`) data, hyper-parameters are transferred to the training part of early past data, which we call Out-of-Sample (`OOS`) data. These hyper-parameters will help to find another set of parameters, called intrinsic-parameters, that are optimized in order to update the ensemble of classifiers to more recent data. Then, any new forecasting method can be tested. This reduces the problem of creating *ad-hoc* ensembles for specific markets, as our ensemble method outputs a pool of best classifiers for any market as soon the market data are in the `IS` and training part of `OOS` sets. Additionally, we allow any number and type of classifiers technologies in the proposed ensemble, minimizing the brute force search for specific classifiers in an ensemble.

Our proposed auto-configurable ensemble is composed of three steps, as follows:

1. **Feature Selection:** data from the target market is pre-processed, with parameters being learned in the IS data.

2. **Two-Step-Auto Adjustable Parameters Ensemble Creation:** with the auto-configurable optimized sets of hyper-parameters and intrinsic-parameters found in IS data, the approach outputs the set of hyper-parameters only, which will be transferred to a new optimization round. This new optimization step is done in the training part of the OOS data, and will find final intrinsic-parameters in recent data to build the final ensemble of classifiers.

3. **Policy for Trading**: we define how to use the created ensemble to trade.

Detailed discussions of these steps are done in the next subsections.

4.1.1 Feature Selection

In order to reduce noise from the data, the literature reports some approaches able to better generalize the involved information by selecting only the characteristics that best represent the domain taken into account (*e.g.*, the stock market). Although other feature selection techniques could be used by our proposed approach, we considered the Independent Component Analysis (ICA) in our approach, as it was, as far as we know, not fully explored in the financial market context. This feature selection approach is able to extract independent streams of data from a dataset composed of several unknown sources, without needing to know any criteria used to join them [115].

The idea of ICA is to project the d dimensional input space into a lower dimensional space d'. This is done by finding a linear representation of non-gaussian data, so the components are statistically independent. Let us assume a d dimensional observation vectors $x = \{x_1, x_2, ..., x_d\}$ composed of zero mean random variables. Let $s = (s_1, s_2, ..., s_d)$ be the d-dimensional transform of x. Then, the problem is to determine a constant weight matrix W so that the linear transformation of the observed variables

$$s = Wx \tag{4.1}$$

has certain properties. This means that the input x can be written in terms of the independent components, or

$$x = A^{-1}s, \tag{4.2}$$

where A is the inverse (or the pseudo-inverse) of the W transform matrix.

The ICA Based dimensionality reduction algorithm is based on the idea that the features that are least important are the ones whose contributions to the independent components are the least. The least important features are then eliminated and the independent components are recalculated based on the remaining features. The degree of contribution of a feature is approximated as the sum of the absolute values of the transform matrix W entries associated with that feature. The ICA process considers the input data as a non-linear combination of independent components by assuming that such a configuration is true in many real-world scenarios, which are characterized by a mixture of many non-linear latent signals [116, 117]. A more rigorous formalization of ICA is provided in [118], where a statistical *latent variables* model has been adopted. It assumes that we observe n linear mixtures of n independent components.

In our optimized ICA approach, we select the best possible number of parameters to be used by this technique, no matter the market considered. This is done by adjusting hyper-parameters, a step further discussed in the next subsection.

4.1.2 Two-Step Auto Adjustable Parameters Ensemble Creation

This Section discusses the proposed method of generating automatically an ensemble of several classifiers to trade in any kind of market. We start by giving an overview of the approach, then we show how we perform optimization of parameters and, finally, we describe the parameters to be learned in order to achieve the final ensemble.

Overview

Our method is a self-configurable ensemble of classifiers whose pipeline can be see in Fig. 4.1. In our approach, hyper-parameters are optimized through performances metrics calculated for the ensemble in the IS data, and are transferred to the training set of OOS (more recent past) data. Finally, intrinsic-parameters are found for the classifiers of the final ensemble, considering more recent past data and the ensemble is updated to test any kind of new data.

The performance metrics we consider in our study lie within the Machine Learning and the economic domains. The rationale behind that is that, in addition to a mere evaluation of the percentage of correct predictions (*i.e.*, accuracy), it is also necessary to estimate the impact of them at the economic level. For instance, the measurement of a good accuracy in the predictions related to a period of five years is not significant if for some intervals of this period (*e.g.,* two consecutive years) we suffered huge economic losses that, certainly, in a real-world scenario, would have stopped any further investment. For this reason, together with the *Accuracy* metric, we adopted as evaluation metrics the *Maximum Drawdown*, the *Coverage*, and the *Return*

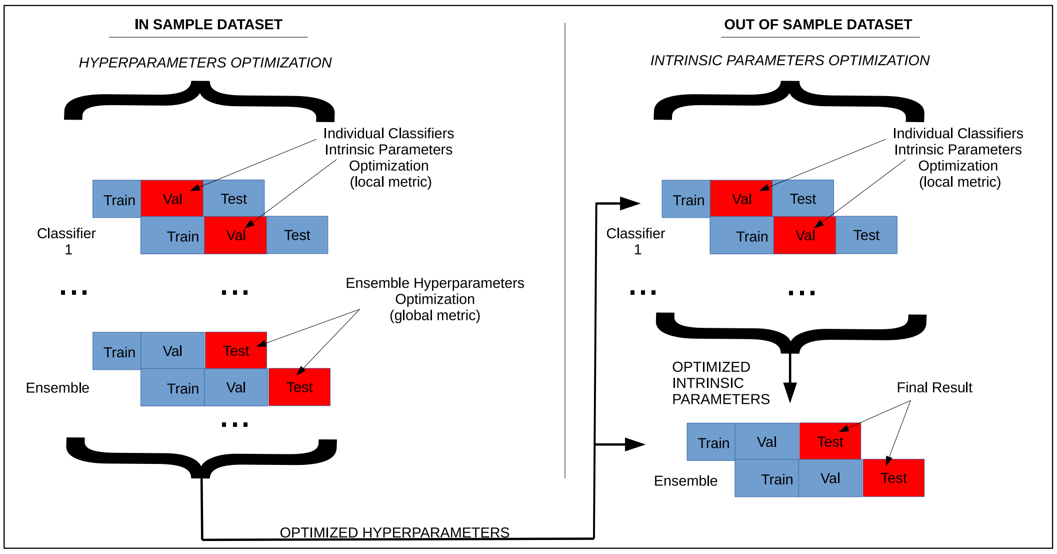

IN SAMPLE DATASET
HYPERPARAMETERS OPTIMIZATION
Individual Classifiers Intrinsic Parameters Optimization (local metric)
Train
Val
Test
Classifier 1
Ensemble Hyperparameters Optimization (global metric)
Ensemble
OPTIMIZED HYPERPARAMETERS
OUT OF SAMPLE DATASET
INTRINSIC PARAMETERS OPTIMIZATION
OPTIMIZED INTRINSIC PARAMETERS
Final Result

Figure 4.1: The proposed two-Step auto adjustable parameters ensemble for market forecasting. This approach optimizes two sets of parameters for a final ensemble to trade in any market. Firstly, in an in-sample dataset with late past data, we optimize the hyper-parameters, considering performance metrics in the testing part of data. Then, these hyper-parameters are transferred to create ensembles for an early past (out of sample) dataset. The individual classifiers have their intrinsic-parameters updated in the validation part of this recent data and then the final ensemble is built.

Over Maximum Drawdown, whose formalization will be provided later in Section 2.6.

To illustrate the benefits of our proposed auto-configurable ensemble method, we build it considering three basic state-of-the-art classifiers [119, 14, 120]: *Gradient Boosting (GB)*, *Support Vector Machines* (*SVM*), and *Random Forests* (*RF*), although any other kinds of classifiers may either replace those or be plugged in. Our method has two sets of parameters to be learned, through a methodology described in details in the next subsection.

Walk-forward Optimization

One of the most used optimization approaches within a Financial Forecasting process for the detection of the best parameters to adopt in the trading strategy is called *Walk Forward Optimization* (*WFO*) [121]. We adopt such a strategy to find the best ensemble hyper-parameters in the `IS` data and intrinsic-parameters in part of `OOS` data. It works by isolating the `IS` time series into several segments, or walks, where each segment is divided in two parts: *Training* and *Testing* sets. The parameters optimization for the used trading strategy is then performed by (i) using several combinations of parameters to train the model in the training part of a segment; and (ii) declare the best (optimized) parameters the ones that yield best performance in the testing set of the same segment. The process is then repeated on the other segments. The performance obtained in the testing set of each segment is not biased as we are not using unknown data, but just `IS` data. The *Walk Forward Optimization* can be performed by following two methodologies, described as follows:

1. Non-anchored Walk Forward Optimization: this approach creates walks of same size. For example, let us assume we have a dataset composed of 200 days that we want to divide into 6 walks of 100 days. One way is to consider the first 80 days of each walk as the training set and the remaining 20 days as the testing set, as shown in the left side of Table 4.1.

2. Anchored Walk Forward Optimization: in this scenario, the starting point of all segments is the same. Additionally, the training set of each segment is longer than the training set of the previous one, therefore the length of each walk is longer than the length of the previous one, as shown in the right side of Table 4.1.

In our approach, we consider the *non-anchored* modality of the *Walk Forward* process, a widely used approach in the literature for Financial Markets [122]. Additionally, the non-anchored WFO used in our approach further subdivides the training data in Table 4.1 into training and validation data,

Table 4.1: Non-anchored and Anchored WFO

Data	Non-anchored WFO			Anchored WFO		
segment/walk	Training	Testing	Days	Training	Testing	Days
1	$1 \rightarrow 80$	$81 \rightarrow 100$	100	$1 \rightarrow 80$	$81 \rightarrow 100$	100
2	$21 \rightarrow 100$	$101 \rightarrow 120$	100	$1 \rightarrow 100$	$101 \rightarrow 120$	120
3	$41 \rightarrow 120$	$121 \rightarrow 140$	100	$1 \rightarrow 120$	$121 \rightarrow 140$	140
4	$61 \rightarrow 140$	$141 \rightarrow 160$	100	$1 \rightarrow 140$	$141 \rightarrow 160$	160
5	$81 \rightarrow 160$	$161 \rightarrow 180$	100	$1 \rightarrow 160$	$161 \rightarrow 180$	180
6	$101 \rightarrow 180$	$181 \rightarrow 200$	100	$1 \rightarrow 180$	$181 \rightarrow 200$	200

Table 4.2: Intra-parameters grid

Algorithm	Parameter	Values	Description
Gradient Boosting	*n_estimators*	$10, 25, 50, 100$	Boosting stages to perform
	learning_rate	$0.0001, 0.001, 0.01, 0.1$	Contribution of each tree
	max_depth	$2, 4, 6, 8, 10$	Maximum depth of each estimator
Support Vector Machines	*max_iter*	$20, 50, 100$	Hard Limit on iterations within solver
	tol	$0.0001, 0.001, 0.01, 0.1$	Tolerance for stopping criterion
	C	$1, 10, 20, 50$	Penalty of the error term
	gamma	$0.0001, 0.001, 0.01, 0.1$	Coefficient for the used kernel
Random Forests	*n_estimators*	$20, 50, 100$	Trees in the forest
	max_depth	$1, 5, 10, 50$	Maximum depth of the tree
	min_samples_split	$0.2, 0.4, 0.8, 1.0$	Minimum samples to split an internal node

where the validation data is 30% of the training data. Then, the performance in the validation data will help finding a set of intrinsic-parameters of the classifiers of the ensemble, whereas the performance in the testing data will find the hyper-parameters of the ensemble. We discuss such auto-configurable parameters in the next section.

Transferable Self Configurable Parameters

With the information of the classifiers used and the optimization methodology in mind, we finally describe the parameters to be found in order to generate the final ensemble. The first set of parameters to be learned through non-anchored WFO comes from the classifiers and are reported in Table 4.2, along with a list of values that must be *grid searched* within the process. Other values and even other parameters can be added too, making the classifiers even more robust to the uncertainties in the training data. Such values are optimized according to the performances in the validation data, a fraction of the *training* data discussed before in section 4.1.2.

The second set of parameters to be tuned is represented by the hyper-parameters, which are not from the classifiers anymore, but are related to the non-anchored WFO and ICA feature selection. Table 4.3 shows the hyper-parameters that need to be optimized according to the chosen metric. They are (i) the dimension of the window for each walk; (ii) the training set size; (iii) lag size; and (iv) number of output signals of the considered `ICA` feature selection approach. These hyper-parameters are optimized through the chosen performance metrics after ensemble classification of *testing* data, where $test\_size = window\_size - train\_size - validation\_size$. This hyper-parameters self-configuration step of our approach is carried out only within the `IS` part of our dataset, according to a considered metric. Once the hyper-parameters and intrinsic-parameters are found in the `IS` data, the algorithm transfers the hyper-parameters only to the `OOS` dataset. Then, only the intrinsic-parameters of the ensemble are optimized and, thus, the ensemble is ready to test new data.

Table 4.3: Hyper-parameters Grid

Parameter	Values	Description
$window\_size$	$100, 150, 200, 250, 300$	Days used for the training and test sets definition
$train\_size$	$60, 65, 70, 75, 80$	Percentage of window_size used for the training set
$lags$	$1, 3, 5, 7, 9$	Previous days to use in order to predict the next one
ica_comp	$1, 3, 5, 7, 9$	Independent Component Analysis output components

Algorithm 1 describes the proposed approach of multi-classifiers auto-configurable ensemble. The algorithm has three main variables: (i) MAX_FINAL_METRIC (initialized in step 7 of the algorithm), which will be used in step 28 to check which hyperparameter $h \in H$ has the best ensemble performance metric; (ii) $ENS\_METRICS$ (initialized in step 8 of the algorithm), which will sum up the metric of applying the ensemble in test part of `IS` data in all walks; and (iii) MAX_WALK_METRIC (initialized in step 14 of the algorithm), which will be used to optimize classifiers intrinsic-parameters in the validation data of each walk. The algorithm starts by, given a combination of hyper-parameters $h \in H$, building the walks W (step 10) and, for each walk $w \in W$, it builds and transforms features (steps 12 and 13), doing grid search in all the classifiers intrinsic-parameters combinations $i \in I$ in order to find the best classifier for each walk (steps 16-22). After the best of each classifier is found for a walk, we apply the ensemble of them accumulating the performance metric in the testing data for all the walks (step 26). After this is done for each hyperparameter combination, we verify, in steps 28-30, if the total metric of the ensemble in all the walks is the highest possible. When all the hyperparameter combinations h have their ensemble tested and with their accumulated metrics on the testing data calculated, in step 33 the algorithm is sure that it found the best possible hyperparameter $h' \leftarrow H$, which is returned by the algorithm.

After the hyper-parameters are found in the `IS` data, we start the search for the intrinsic-parameters of the ensemble in recent past data, and then our ensemble is ready and can already trade. Such procedure is reported in Algorithm 2. In this algorithm, just two metrics are necessary: (i) the variable MAX_WALK_METRIC (step 12 of the algorithm) to tune the intrinsic-parameters of the classifiers in the new `OOS` data; and (ii) $METRIC$ (step 8 of the algorithm) to calculate the final metrics of the ensemble trading on unseen `OOS` data. The process is similar to Algorithm 1, with the difference being the fact that hyper-parameters are not searched anymore and the testing data is used to report trading real-time results. The algorithm returns the mean metric, considering the whole testing period.

Doing the search of parameters this way, the hyper-parameters of the final ensemble will be optimized in the `IS` data through non-anchored WFO. Then, these hyper-parameters are transferred to the non-anchored WFO of the `OOS` data, and intrinsic-parameters are now optimized in the validation data only. Thus, an auto adjustable ensemble approach is built in such a way that will return an ensemble of the best possible classifiers for any market, as long as their `IS` and training and validation `OOS` data are fed to the algorithm, being this way a *data-driven* optimization approach.

4.1.3 Policy for Trading

Many literature studies [123, 124] demonstrate the effectiveness of ensemble approaches that implement different algorithms and feature subsets. Ensemble approaches [125] usually get the best results in many prestigious Machine Learning competitions (*e.g.*, Kaggle, Netflix Competition, KDD, and so on).

Therefore, in this work, we are adopting an *ensemble learning* approach, which means that the final result (*i.e.*, the prediction) is obtained by combining the outputs made by single algorithms in the ensemble. As stated before in Chapter 3, such an ensemble process can work in a dependent or independent fashion. The approach we choose is the independent framework, so each classifier decision may represent a vote that is independent from the others. We apply such an approach using three selected algorithms (*i.e.*, *Gradient Boosting*, *Support Vector Machines*, and *Random Forests*) with their ensemble hyper-parameters initially found in the `IS` data, and whose individual classifiers intrinsic-parameters are found in the `OOS` data. We adopt in our ensemble approach the aggregation criterion called *complete agreement*, meaning that we make our prediction to do a*long* or *short* operation only if there is a total agreement among all the algorithm predictions, otherwise we do not make a prediction for the related futures market (*flat*). This is an approach that usually leads towards better predictive performance, compared to that of each single algorithm. Such an approach for the future day prediction is better illustrated in Algorithm 3.

Algorithm 1 Proposed hyperparameter search approach

```
1.22
Require:
  $IS$=time series from in sample data
  $I$= list of intra-parameters as shown in Table 4.2
  $H$=list of hyper-parameters as shown in Table 4.3
  $C$=list of classifiers from the ensemble
Ensure:
  $h'$= Optimized hyper-parameters
  procedure RETURN_HYPER-PARAMETERS($IS$, $I$, $H$, $C$)
      $MAX\_FINAL\_METRIC \leftarrow 0$
      $ENS\_METRIC \leftarrow 0$
      for $h$ in $H$ do                                ▷ for each hyperparameter combination
          $W[h] \leftarrow buildWalks(IS, h(window\_size))$    ▷ Starts non-anchored WF0
          for $w$ in $W[h]$ do                                            ▷ for each walk
              $F \leftarrow buildFeatures(w, h(lags))$                      ▷ get features
              $F' \leftarrow icaTransform(h(ica\_comp), F)$            ▷ transform features
              $MAX\_WALK\_METRIC \leftarrow 0$
              for $c$ in $C$ do                                         ▷ for each classifier
                  for $i$ in $I$ do         ▷ for each intrinsic parameter, train and validate
                      $M[i] \leftarrow trainClassifier(F', h(train\_size), c[i])$
                      $METRIC \leftarrow testClassifier(M[i], F'[h(train\_size) * 0.3])$
                      if $METRIC > MAX\_WALK\_METRIC$ then
                          $E[c, w] \leftarrow M[i]$
                          $MAX\_WALK\_METRIC \leftarrow METRIC$
                      end if
                  end for
              end for
              $test\_data \leftarrow F'[h(window\_size) - h(train\_size) - h(train\_size) * 0.3]$
              $ENS\_METRIC \leftarrow ENS\_METRIC + testClassifier(E[C, w], test\_data)$
          end for
          if $ENS\_METRIC > MAX\_FINAL\_METRIC$ then
              $h' \leftarrow h$
              $MAX\_FINAL\_METRIC \leftarrow ENS\_METRIC$
          end if
      end for
      return $h'$
  end procedure
```

Algorithm 2 Proposed intrinsic parameter search approach and ensemble trading

```
1.22
Require:
    $OOS$=time series from in sample data
    $I$= list of intra-parameters as shown in Table 4.2
    $h'$=best hyperparameter found in Algorithm 1
    $C$=list of classifiers from the ensemble
Ensure:
    $MEAN\_METRIC$= Mean performance of trading
    procedure ENSEMBLE_TRADING($OOS$, $I$, $h'$, $C$)
        $W \leftarrow buildWalks(OOS, h'(window\_size))$    ▷ Starts non-anchored WFO
        $METRIC \leftarrow 0$    ▷ Metric used to report testing results
        for $w$ in $W$ do    ▷ for each walk
            $F \leftarrow buildFeatures(w, h'(lags))$    ▷ get features
            $F' \leftarrow icaTransform(h'(ica\_comp), F)$    ▷ transform features
            $MAX\_WALK\_METRIC \leftarrow 0$
            for $c$ in $C$ do    ▷ for each classifier
                for $i$ in $I$ do ▷ for each intrinsic parameter, train and validate
                    $M[i] \leftarrow trainClassifier(F', h'(train\_size), c[i])$
                    $METRIC \leftarrow testClassifier(M[i], F'[h'(train\_size) * 0.3])$
                    if $METRIC > MAX\_WALK\_METRIC$ then
                        $E[c, w] \leftarrow M[i]$
                        $MAX\_WALK\_METRIC \leftarrow METRIC$
                    end if
                end for
            end for
            $test\_data \leftarrow F'[h'(window\_size) - h'(train\_size) - h'(train\_size) * 0.3]$
            $METRIC \leftarrow METRIC + testClassifier(E[C, w], test\_data)$
        end for
        $MEAN\_METRIC \leftarrow METRIC/|W|$
        return $MEAN\_METRIC$
    end procedure
```

Algorithm 3 *Future day prediction*

```
1.3
Require: $A$=Set of algorithms, $D$=Past classified trading days, $\widehat{d}$=Day to predict
Ensure: $result$=Day $\widehat{d}$ prediction
 1: procedure PREDICTION($A$, $D$, $\widehat{d}$)
 2:     $models = trainingModels(A, D)$
 3:     $predictions = getPredictions(models, \widehat{d})$
 4:     if $agreement(predictions) == TRUE \wedge predictions == -1$ then
 5:         $result \leftarrow short$
 6:     else if $agreement(predictions) == TRUE \wedge predictions == 1$ then
 7:         $result \leftarrow long$
 8:     else
 9:         $result \leftarrow flat$
10:     end if
11:     return $result$
12: end procedure
```

4.2 Experimental Setup

In this section, we discuss the setup chosen to guide the experiments performed to validate our ensemble approach against some benchmarks from the literature. We start discussing the datasets, the performance metrics and implementation aspects of our proposed method and of the benchmarks.

4.2.1 Formal Notation

We denote a set of data composed by a series of consecutive trading days of a *futures market* $D = \{d_1, d_2, \ldots, d_N\}$, and a set of features $F = \{date, open, high, low, close, volume, next\}$ that compose each $d \in D$[1], where $next = 1$ if the $close - open$ of the next day is ≥ 0, otherwise $next = -1$ (we also associated the meaning of *long* operation to 1 and the meaning of *short* operation to -1[2]).

We denote as $D_+ \subseteq D$ the subset of trading days with a positive closing (i.e., $close - open \geq 0$), and as $D_- \subseteq D$ the subset of trading days with a negative closing (i.e., $close - open < 0$).

We denote as $\widehat{D} = \{\widehat{d}_1, \widehat{d}_2, \ldots, \widehat{d}_U\}$ a set of unclassified trading days. It should be observed that, according to the aforementioned definition of $next \in$

[1]They stand for *date of day*, *open value*, *highest value reached in the day*, *lowest value reached in the day*, *close value*, and *exchange volume*.

[2]They represent the operations allowed on the futures markets taken into consideration in this paper.

F, a trading day can only belong to one class $c \in C$, where $C = \{1, -1\}$.

We also denote as $I = \{i_1, i_2, \dots, i_X\}$ the components of each trading day $d \in D$, obtained by transforming the original data through a feature selection process, which in our case is *ICA*.

Finally, we denote a set of operations $O = \{long, short, flat\}$ allowed on a futures market, where *flat* means that no operation of *long* or *short* has been performed [3].

4.2.2 Datasets

To verify our approach performance against some benchmarks, we selected four datasets based on stock futures markets (*SP500*, *DAX* and *FIB*) and one future of commodity (*CL*). As far as the stock futures markets are concerned, we included the *FIB* market, which is characterized by an atypical behavior with respect to the other stock futures markets in the years taken into account during the experiments. We based our choice on the observation that stock markets behavior is usually different from that of the bond markets, as there usually exists an inverse correlation between them. Indeed, the stock futures are frequently characterized by a strong upward bias (*e.g.*, *SP500* and *DAX*), with some exceptions related to some particular economic scenarios, as it happened for the Italian *FIB* in recent years. Details of such datasets are reported in Table 4.4.

Table 4.4: Futures Market Datasets

Futures dataset	Name	From day	To day	Trading days
*SP*500	Standard & Poors 500	02/01/2008	31/12/2018	2827
DAX	German Market	02/01/2008	28/12/2018	2792
FIB	Italian Market FIB Future	02/01/2008	27/12/2018	2790
CL	Light Sweet Crude Oil Future	02/01/2008	28/12/2018	2774

These datasets can be easily found at different time resolutions (*e.g.*, *5-minutes*, *10-minutes*, *1-hour*, *etc.*). In this work, we further transform the futures market datasets by adopting a *1-day* resolution. It means that, starting from the original resolution that characterizes the dataset (*e.g.*, *5-minutes*, *10-minutes*, *1-hour*, among others), the data have been opportunely joined in order to obtain for each day included in the dataset the following new information *I*={*date, open value, highest value, lowest value, close value, exchange volume*}, where each record of the new dataset corresponds to one day. As the *SP500* market has a point value of *50 USD*, the *DAX* market has a point value of *25 EURO*, the *FIB* market has a point value of *5 EURO* and the *CL*

[3] https://www.thebalance.com/long-and-short-trading-term-definitions-1031122

market has a point value of *1,000 USD*, in order to simplify, we do not convert the points to their corresponding currency values, keeping such information in points.

In these datasets, we denote a set of data composed of a series of consecutive trading days as $X = \{x_1, x_2, \dots, x_N\}$, and a set of features $F = \{date, open, high, low, close, volume, next\}$ that compose each $x \in X$ and $next = 1$ if the $close - open$ of the next day is greater than or equal to zero, and $next = -1$ otherwise. We also label the *long* operation to 1, and *short* operation to -1, as they represent the operations allowed on the futures markets taken into consideration in this paper.

It should be observed that, according to the aforementioned definition of $next \in F$, a trading day can only belong to one class $c \in C$, where $C = \{1, -1\}$. We also denote as $I = \{i_1, i_2, \dots, i_X\}$ the components of each trading day $x \in X$, obtained by transforming the original data through a feature selection process, which in our case is *ICA*. Finally, we denote a set of of operations $O = \{long, short, flat\}$ allowed on a futures market, where $flat$ means that no operation of *long* or *short* has been performed.

Given the previous definitions, for each trading day x (*i.e.*, each dataset row), we add a further data field *next*, which corresponds to the target class related to the next trading day $x+1$, and is defined according to the notation reported in Equation 4.3.

$$next_x = \begin{cases} \textit{1}, & \text{if } (close_{x+1} - open_{x+1}) \geq 0 \\ \textit{-1}, & \text{otherwise.} \end{cases} \tag{4.3}$$

We let the reader observe that the time series resolution may be set even to a finer scale, *e.g.*, hours or minutes. In such a case, a record in a given time interval would consist of a group features {*time, open value, highest value, lowest value, close value*} for each considered interval, ended with the *next* class, as defined above.

To train our prediction models with more than a day of the features market (*lags* hyper-parameter in Table 4.3), we can arbitrarily aggregate more days, obtaining a series of vectors V composed of `ICA` components of N days, with the *next* value as target class. As an example, assuming we have to aggregate three days x_1, x_2, x_3, each of them characterized by two `ICA` components i_1, i_2, we would obtain the vector shown in Equation 4.4.

$$V = [x_1(i_1),\ x_1(i_2),\ x_2(i_1),\ x_2(i_2),\ x_3(i_1),\ x_3(i_2),\ next_{x_4}] \tag{4.4}$$

In our experiments, we report the experiments considering the period from 2016 to 2018 as `OOS` data, where we have updated and tested our approach after the auto-configuration and tuning of the hyper-parameters in the `IS` data, which uses the remaining years.

4.2.3 Technical Details

The approach proposed in this paper has been developed in *Python*, as well as the implementation of the state-of-the-art classification techniques used to define our ensemble approach, which are based on *scikit-learn*[4]. In order to make our experimental results reproducible, we have set to zero the seed of the pseudo-random number generator used by the *scikit-learn* evaluation algorithms. The PC where all the experiments have been performed is an *Intel i7-3770S, octa-core* (*3.10 GHz* $\times$ *8*) with a *Linux 64-bit* Operating System (*Debian Stretch*) with *8 GBytes* of *RAM*.

As for the benchmarks, we firstly considered the common Buy and Hold benchmark strategy. It represents a passive investment strategy in which the investors buy futures and hold them for a long period, regardless of the market's fluctuation. Such a strategy is largely used in literature as a benchmark to evaluate the profitability of an investment strategy. In addition, we performed the future market predictions by using single predictors (*i.e.*, *GB*, *SVM*, and *RF*), configuring their default hyper-parameters according to some common values in the literature: 40% of the `IS` dataset as walk size, of which 75% is used as training set and the remaining 25% as validation set with 5 day-lags [34, 126, 35, 127, 128]. Finally, we also used a recent approach to perform trading [81], which we call *TSVR* in the remaining of this paper. For this approach, we used both the linear and nonlinear kernel. As described in in [81], we have used 10-fold cross validation in the training data to find the kernel parameters that yielded the best mean squared error in all markets. As this approach is proposing to predict the closing price (regression problem), we mapped the problem consistently with ours and changed the output so that for each day we have either a long or short operation. As with our approach, final results are reported in terms of classification performance in the testing part of the `OOS` dataset.

Regarding time consumption related to our approach, we can observe from the pipeline showed in Fig. 4.1 that it is strictly related to the canonical time spent by each algorithm that composes the ensemble, multiplied by the intrinsic-parameters involved in the auto-tuning process plus the time spent by other processes (*i.e.*, *walk-forward* and *ICA Feature Selection*), since the detection process of the optimal hyper-parameters has been previously (one time) performed in the in-sample part of the datasets, therefore it does not need to be repeated at each prediction.

More formally, assuming t being the execution time of each ensemble algorithm, na the number of algorithms in the ensemble, np the number of parameters involved in the auto-tuning process and Δ the execution time related to the other processes, the total time consumption τ can be formalized as shown in Equation 4.5.

[4]`http://scikit-learn.org`

$$\tau = (t \cdot na \cdot np) + \Delta \tag{4.5}$$

For example, with a previous information that the proposed approach involves 3 algorithms (*i.e.*, *Gradient Boosting*, *Support Vector Machine*, and *Random Forests*) with, respectively, 3, 4, and 3 intrinsic-parameters, and by using a machine with the software and hardware characteristics reported in Section 4.2.3, the average time consumption for each prediction on the markets taken into account is reported in Table 4.5.

Table 4.5: Time Consumption

Futures market	Average prediction time in seconds
SP500	5.36
DAX	1.69
FIB	5.40
CL	13.57
Mean time	6.50

It should be observed that such a running time can be effectively reduced by parallelism of the process over several machines, both along algorithms and markets, by exploiting large scale distributed computing models such as MapReduce [129, 130]. This improves the approach scalability in the context of applications that deal with frequency trading.

4.3 Experiments

In this section, we discuss the experimental results of applying our approach in four different markets, comparing it with common benchmarks and state-of-the-art approaches. We divide this section into two subsections: firstly, we start in Section 4.3.1 the discussion of results in a natural trading setup. Then, we perform in Section 4.3.2 a study of the impact of different performance metrics in the proposed trading system.

4.3.1 Trading Results

We firstly start showing trading results of the non-optimized individual classifiers and other benchmark approaches considered in the experiments in Table 4.6, where *BH* stands for *Buy and Hold*, *GB* stands for *Gradient Boosting*, *SVM* stands for *Support Vector Machines*, and *RF* stands for *Random Forests*. All the values are expressed in futures market points, with the exception of those expressed as a percentage. It should be noted that all the experiments have been performed by taking into account the same *out-of-sample* time period used for the performance evaluation of our approach (*i.e.*, years from 2016 to 2018).

It can be seen from Table 4.6 that the Buy and Hold strategy performed well in two (*SP500* and *DAX*) out of four markets in terms of return if compared to benchmark approaches. Regarding the individual classifiers, the best mean accuracy of 53.75% was achieved by the *SVM* classifier. Such a high mean accuracy makes this classifier achieving the highest return in *FIB* market. According to *MDD*, it is less risky (lowest MDD) for *FIB* and *SP500* markets. The *GB* classifier got the second place with 50.50% mean accuracy. However, it is not the best trading strategy in terms of return and risk in any of the markets considered, which allows us to reach an interesting conclusion that higher accuracies do not necessary imply better trading strategies. *RF* classifier achieved a 50% mean accuracy. In terms of risk, it outperforms others in *DAX* market. Finally, *TSVR* showed a very poor algorithm performance. There are two possible explanations for its poor behavior in all these markets in general: (i) TSVR is trained to perform regression instead of classification (we use the sign of predictor to do a decision); and (ii) we consider the standard parameters of such a technique. With such findings, we enforce the necessity of using optimized parameters and ensembles of classifiers with diverse behaviors.

Table 4.6: Buy and Hold and Single Predictors Performance with Default Configurations

Strategy	Market	Accuracy	MDD	MDD%	Return	Return%	Romad
BH	SP500	–	601.75	29.53	473	23.21	0.79
BH	DAX	–	3102.5	29.49	51.5	0.49	0.02
BH	CL	–	34.29	65.71	-5.92	-11.35	-0.17
BH	FIB	–	6175	29.25	-3135	-14.85	-0.51
GB	SP500	0.51	465.75	22.86	136.5	6.7	0.29
GB	DAX	0.49	4970.5	47.25	-3878	-36.86	-0.78
GB	CL	0.52	27.55	52.8	16.39	31.41	0.59
GB	FIB	0.5	7060	33.44	-2843	-13.47	-0.4
SVM	SP500	0.55	401.25	19.69	359	17.62	0.89
SVM	DAX	0.52	2901	27.58	-1040	-9.89	-0.36
SVM	CL	0.54	26.47	50.73	18.83	36.09	0.71
SVM	FIB	0.54	4640	21.98	11483	54.4	2.47
RF	SP500	0.47	587.75	28.84	-317.5	-15.58	-0.54
RF	DAX	0.5	2516	23.92	-1329	-12.63	-0.53
RF	CL	0.52	20.79	39.84	34.17	65.48	1.64
RF	FIB	0.51	9133	43.26	2237	10.6	0.24
TSVR_LIN	SP500	0.21	4807.75	235.93	-2929	-143.74	-0.61
TSVR_LIN	DAX	0.4	18386.5	174.78	-3456	-32.85	-0.19
TSVR_LIN	CL	0.37	201.53	386.22	-111.25	-213.2	-0.55
TSVR_LIN	FIB	0.48	41280	195.55	8503	40.28	0.21
TSVR_NONLIN	SP500	0.22	4445.25	218.15	-2566.5	-125.95	-0.58
TSVR_NONLIN	DAX	0.4	17421.5	165.61	-2734	-25.99	-0.16
TSVR_NONLIN	CL	0.39	180.86	346.61	-91.49	-175.34	-0.51
TSVR_NONLIN	FIB	0.47	41660	197.35	7483	35.45	0.18

As next step, we evaluated the performance of the ensembles of each benchmark predictor after we performed the self-configuration step, setting as optimization metric the *Accuracy* as set up in Algorithms 1 and 2. The results related to this experiment are shown in Table 4.7 and consider the optimized individual classifiers versus the benchmarks again. This table shows the benefits of the proposed adjustable parameters optimization approach (intrinsic-parameters and hyper-parameters), where, for example, for the *GB* classifier, accuracy increased in classifying three markets, keeping the same accuracy for the other market. It can also be seen that the *MDD* decreased for some markets.

Finally, the performances of our ensemble approach are reported in Table 4.8. Considering that our ensemble makes its prediction only when all the predictors agree (*complete agreement* strategy), the *Coverage* value indicates the percentage of days when we operated in the futures market (*i.e.*, by placing *Short* or *Long* operations). This means that in the remaining days we do not perform any *Long* or *Short* operations (*i.e.*, we performed *Flat* operations only). The reader may observe that the results obtained by our ensemble approach are more robust with respect to those of single predictors in most of the markets, specially because it is the only technique that yields positive returns for all the markets. Moreover, it can be seen in Table 4.8 an improvement of the proposed approach regarding the benchmarks and optimized classifiers, specially in terms of risk-base metrics (*MDD* and *Romad*). This highlights the benefits of the proposed approach, which differs from the ones of the literature by its total *data-driven* nature, where a bunch of procedures as features selection, parameters tuning and time series adjustments are automatically chosen based on data performance, maximizing ensembles to do conservative investments with minimum losses. This means that the proposed approach can be regarded as a trading strategy useful for novice traders, who are initially careful about their investments.

On the basis of the aforementioned considerations, Fig. 4.3 shows the equity curves for the considered markets. As far as our ensemble approach is concerned, results and discussions are consistent with those in Table 4.8, whereas Buy and Hold results and discussions are consistent with Tables 4.6 or 4.7. The difference is that the equity curve indicates the cumulative return of the underlying market within the considered OOS test data, whereas the results of the tables indicate the final value over the entire period. As far as the *DAX* market is concerned, the reader may notice that, although our proposed ensemble is slightly worse than the results of the *BH* approach, it shows positive return. The worse performance of our approach relies on the fact that we are maintaining the same classifiers in the ensemble, no matter the market. We envision that a further step of the proposed approach in the future is also performing a *data-driven* selection of classifiers in the ensemble according to the market behavior.

Table 4.7: Single Predictors Performance After a Tuning Process Optimized by Accuracy

Strategy	Market	Accuracy	MDD	MDD%	Return	Return%	Romad
BH	SP500	–	601.75	29.53	473	23.21	0.79
BH	DAX	–	3102.5	29.49	51.5	0.49	0.02
BH	CL	–	34.29	65.71	-5.92	-11.35	-0.17
BH	FIB	–	6175	29.25	-3135	-14.85	-0.51
GB	SP500	0.52	704.75	34.58	-309.5	-15.19	-0.44
GB	DAX	0.51	4890.5	46.49	-2587	-24.59	-0.53
GB	CL	0.54	24.13	46.24	9.81	18.8	0.41
GB	FIB	0.5	12553	59.46	-2733	-12.95	-0.22
SVM	SP500	0.55	544.25	26.71	155	7.61	0.28
SVM	DAX	0.51	4033	38.34	-2546	-24.2	-0.63
SVM	CL	0.54	31.84	61.02	-4.97	-9.52	-0.16
SVM	FIB	0.51	7245	34.32	3357	15.9	0.46
RF	SP500	0.51	624.5	30.65	-500	-24.54	-0.8
RF	DAX	0.51	4645.5	44.16	-3720	-35.36	-0.8
RF	CL	0.5	23.82	45.65	11.39	21.83	0.48
RF	FIB	0.5	7358	34.86	-553	-2.62	-0.08
TSVR_LIN	SP500	0.21	4807.75	235.93	-2929	-143.74	-0.61
TSVR_LIN	DAX	0.4	18386.5	174.78	-3456	-32.85	-0.19
TSVR_LIN	CL	0.37	201.53	386.22	-111.25	-213.2	-0.55
TSVR_LIN	FIB	0.48	41280	195.55	8503	40.28	0.21
TSVR_NONLIN	SP500	0.22	4445.25	218.15	-2566.5	-125.95	-0.58
TSVR_NONLIN	DAX	0.4	17421.5	165.61	-2734	-25.99	-0.16
TSVR_NONLIN	CL	0.39	180.86	346.61	-91.49	-175.34	-0.51
TSVR_NONLIN	FIB	0.47	41660	197.35	7483	35.45	0.18

Table 4.8: Ensemble Predictors Performance After a Tuning Process Optimized by Accuracy

Strategy	Market	Coverage	Accuracy	MDD	MDD%	Return	Return%	Romad
ENS	$SP500$	0.63	0.57	406.5	19.95	485.75	23.84	1.19
ENS	DAX	0.55	0.52	2184.5	20.77	9.5	0.09	0
ENS	CL	0.57	0.55	17.56	33.65	13.19	25.28	0.75
ENS	FIB	0.42	0.52	1655	7.84	2760	13.07	1.67

4.3.2 Performance Metrics Trade Impact

In this part of the experiments, we report the trading results of our approach against the Buy and Hold benchmark when considering different performance metrics in Algorithms 1 and 2. Table 4.9 shows the results, where we indicate, for each market, the best hyperparameters found for the ensemble (in blue), classification metrics (in white) and risk and financial metrics (in green), together with BH metrics for the same markets (in gray). With such a study, we investigate what is the best metric to consider when optimizing the intrinsic-parameters and hyper-parameters of the proposed approach.

It can be seen from Table 4.9 that for Accuracy and Return performance metrics, the risk is decreased (better *MDD*) no matter the market used. How-

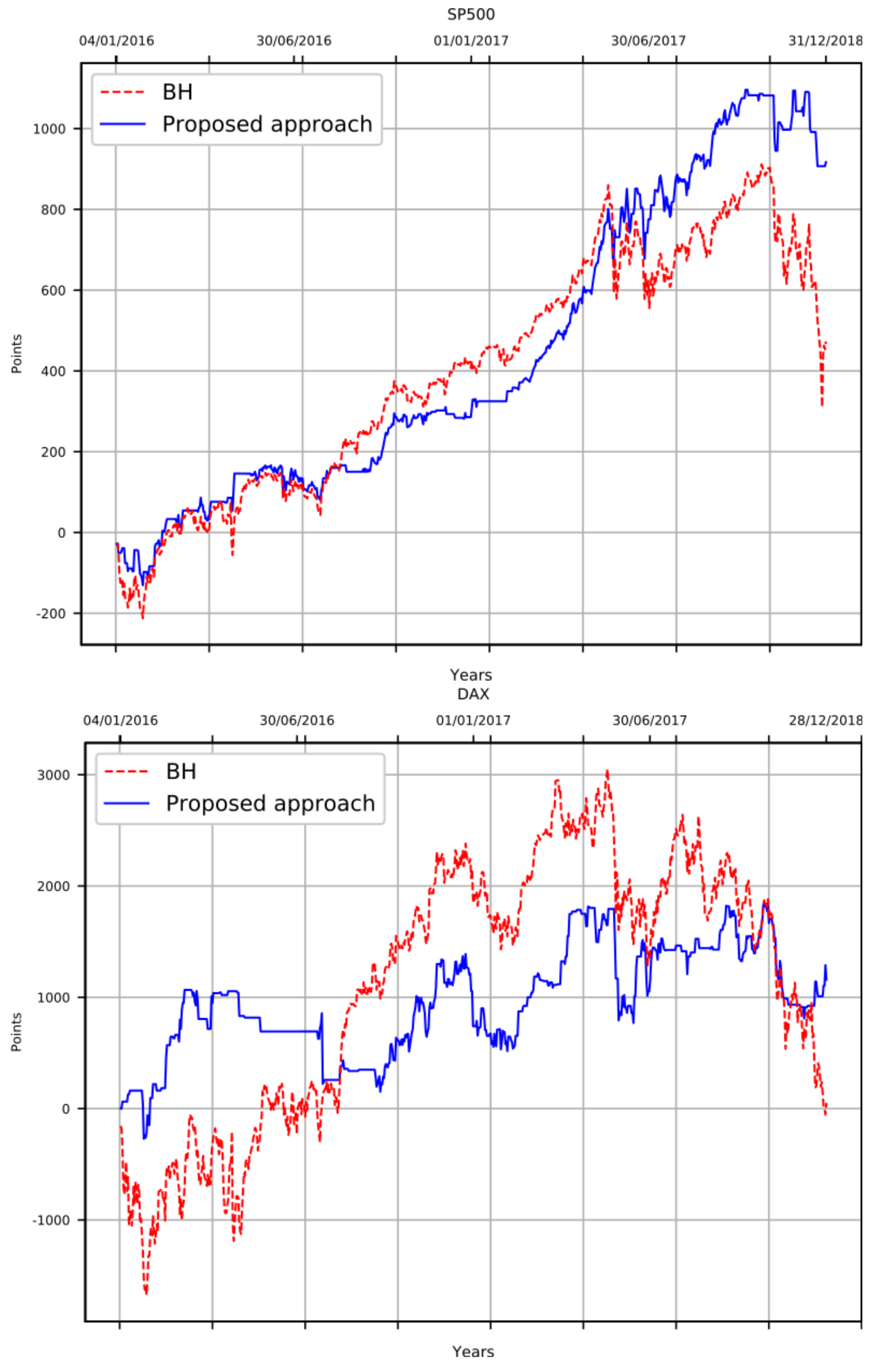


Figure 4.2: $Equity Curves for SP500 \& DAX$

ever, the Accuracy showed to be the best metric as we have four better *MDD* metrics, three better returns and three better *Romads*, totalling ten wins against two losses from *BH*. All the other metrics do not beat the *BH* more

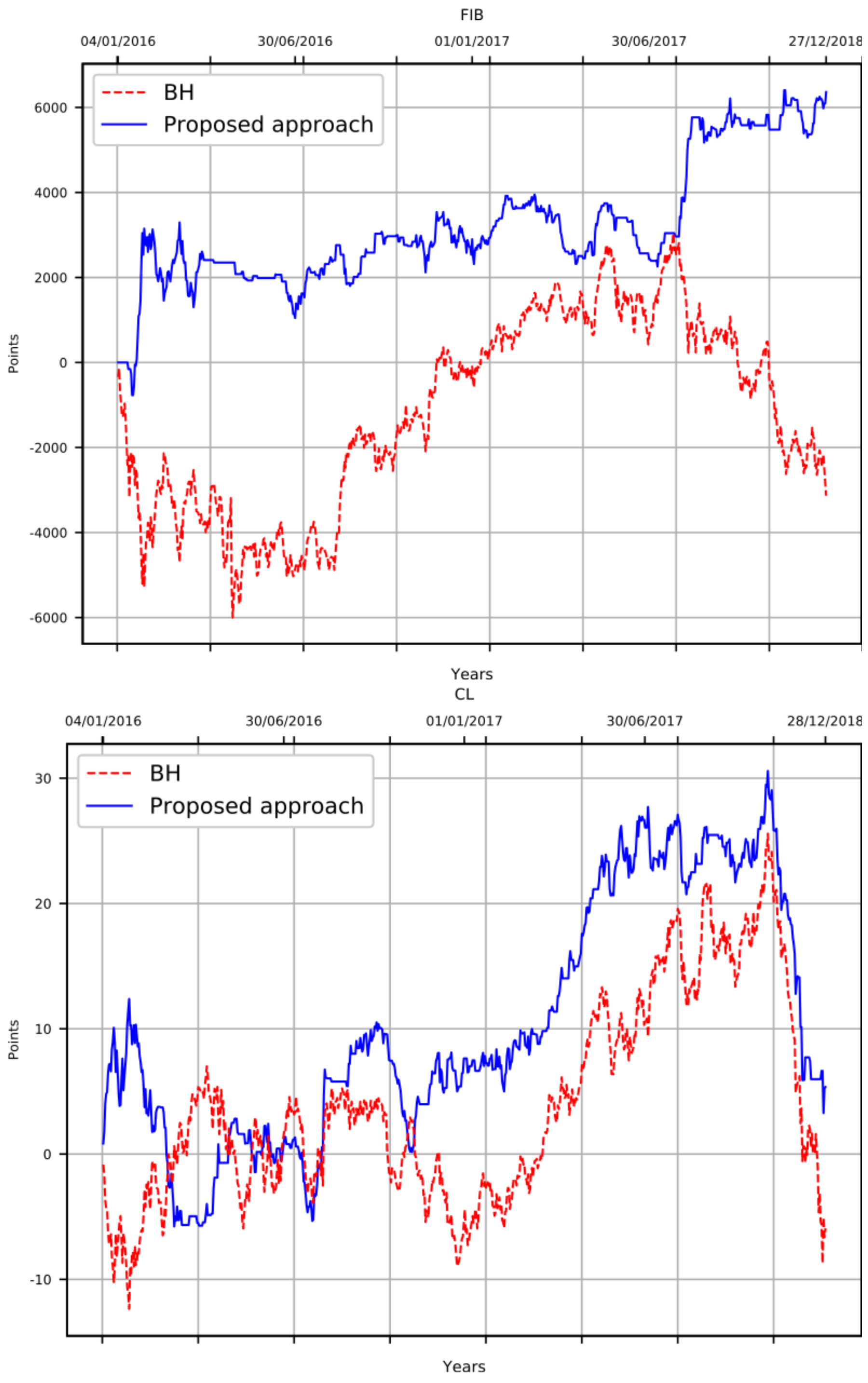


Figure 4.3: *Equity Curves for FIB & CL*

than the accuracy, but the results in this table shows that there is a correlation between high accuracy in the training of the ensemble and a low risk in the trading real-world environment. This can be particularly noted from the results in Table 4.8, where our proposed approach showed to be the only one who got positive returns in all the markets. A solution like the proposed ensemble can be a good solution for initial investors who want a trade-off between low risk and non negative returns. This is explained because an ensemble does not trade all the time (low coverage), so a better accuracy in these less frequent trading times makes a more efficient and less risky trading system. These results are therefore a further evidence of the goodness and robustness of the proposed approach, specially in the presence of atypical and unpredictable markets.

Table 4.9: Comparison of the proposed approach against the Buy and Hold benchmark by considering different performance metrics in building the proposed self-configurable ensemble.

Accuracy														Buy and Hold					Our Performance		
market	year	walk_size	train(%)	lags	icacomp	preacc	accuracy	coverage	mdd	mdd(%)	return	return(%)	romad	mdd	mdd(%)	return	return(%)	romad	mdd(%)	return(%)	romad
SP500	2016-2017-2018	100	65	9	5	0.54	0.57	0.63	406.5	19.95	485.75	23.84	1.19	601.75	29.53	473	23.21	0.79	+	+	+
DAX	2016-2017-2018	300	65	7	0	0.53	0.52	0.55	2184.5	20.77	9.5	0.09	0	3102.5	29.49	51.5	0.49	0.02	+		
CL	2016-2017-2018	200	80	7	5	0.5	0.55	0.57	17.56	33.65	13.19	25.28	0.75	34.29	65.71	-5.92	-11.35	-0.17	+	+	+
FIB	2016-2017-2018	250	65	1	5	0.5	0.52	0.42	1655	7.84	2760	13.07	1.67	6175	29.25	-3135	-14.85	-0.51	+	+	+

Return														Buy and Hold					Our Performance		
market	year	walk_size	train(%)	lags	icacomp	preacc	accuracy	coverage	mdd	mdd(%)	return	return(%)	romad	mdd	mdd(%)	return	return(%)	romad	mdd(%)	return(%)	romad
SP500	2016-2017-2018	300	80	1	3	0.57	0.55	0.54	452.25	22.19	220.5	10.82	0.49	601.75	29.53	473	23.21	0.79	+		
DAX	2016-2017-2018	300	70	3	5	0.55	0.52	0.6	2732	25.97	106	1.01	0.04	3102.5	29.49	51.5	0.49	0.02	+	+	+
CL	2016-2017-2018	250	80	3	5	0.51	0.54	0.55	29.87	57.24	-10.09	-19.34	-0.34	34.29	65.71	-5.92	-11.35	-0.17	+		
FIB	2016-2017-2018	100	80	7	2	0.51	0.52	0.47	2495	11.82	4485	21.25	1.8	6175	29.25	-3135	-14.85	-0.51	+	+	+

RoMaD														Buy and Hold					Our Performance		
market	year	walk_size	train(%)	lags	icacomp	preacc	accuracy	coverage	mdd	mdd(%)	return	return(%)	romad	mdd	mdd(%)	return	return(%)	romad	mdd(%)	return(%)	romad
SP500	2016-2017-2018	200	80	5	5	0.53	0.57	0.68	365.25	17.92	320.5	15.73	0.88	601.75	29.53	473	23.21	0.79	+		+
DAX	2016-2017-2018	200	60	9	5	0.52	0.56	0.42	972	9.24	2400	22.81	2.47	3102.5	29.49	51.5	0.49	0.02	+	+	+
CL	2016-2017-2018	250	80	3	5	0.51	0.54	0.55	29.87	57.24	-10.09	-19.34	-0.34	34.29	65.71	-5.92	-11.35	-0.17	+		
FIB	2016-2017-2018	150	70	3	5	0.49	0.49	0.48	6975	33.04	-5930	-28.09	-0.85	6175	29.25	-3135	-14.85	-0.51			

Return (%)														Buy and Hold					Our Performance		
market	year	walk_size	train(%)	lags	icacomp	preacc	accuracy	coverage	mdd	mdd(%)	return	return(%)	romad	mdd	mdd(%)	return	return(%)	romad	mdd(%)	return(%)	romad
SP500	2016-2017-2018	300	80	1	3	0.57	0.55	0.54	452.25	22.19	220.5	10.82	0.49	601.75	29.53	473	23.21	0.79	+		
DAX	2016-2017-2018	300	70	3	5	0.55	0.52	0.6	2732	25.97	106	1.01	0.04	3102.5	29.49	51.5	0.49	0.02	+	+	+
CL	2016-2017-2018	250	80	3	5	0.51	0.54	0.55	29.87	57.24	-10.09	-19.34	-0.34	34.29	65.71	-5.92	-11.35	-0.17	+		
FIB	2016-2017-2018	100	80	7	2	0.51	0.52	0.47	2495	11.82	4485	21.25	1.8	6175	29.25	-3135	-14.85	-0.51	+	+	+

MDD (%)														Buy and Hold					Our Performance		
market	year	walk_size	train(%)	lags	icacomp	preacc	accuracy	coverage	mdd	mdd(%)	return	return(%)	romad	mdd	mdd(%)	return	return(%)	romad	mdd(%)	return(%)	romad
SP500	2016-2017-2018	250	65	9	2	0.54	0.56	0.56	315.25	15.47	156.5	7.68	0.5	601.75	29.53	473	23.21	0.79	+		
DAX	2016-2017-2018	150	65	1	4	0.5	0.53	0.5	3202	30.44	-2368.5	-22.52	-0.74	3102.5	29.49	51.5	0.49	0.02			
CL	2016-2017-2018	250	65	5	2	0.49	0.51	0.53	24.11	46.21	-5.47	-10.48	-0.23	34.29	65.71	-5.92	-11.35	-0.17	+	+	
FIB	2016-2017-2018	200	80	3	0	0.49	0.47	0.41	5653	26.78	-3753	-17.78	-0.66	6175	29.25	-3135	-14.85	-0.51	+		

Chapter 5

Deep Learning and Time Series-to-Image Encoding

5.1 The proposed approach

As pointed in Chapter 1, most of the research in market prediction and Financial Forecasting is based on Artificial Neural Networks or Machine Learning approaches. These models are commonly trained on time series data describing a market index in the past, with the goal of predicting its future trend. The aim of this work is to achieve market prediction over the S&P500 index, by using an ensemble of CNNs, with the training phase executed over GAF images (particularly, the GADF). This particular kind of imaging technique is detailed in section 5.1.2. The rightmost part of Figure 5.3 shows how the proposed trading system works; firstly, the data of the original time series are aggregated according to 4 intervals of time; then, consecutive time frames of 20 observations are extracted from each time series, in order to generate the related set of GADF images. Twenty similar CNNs (whose architecture is shown in the leftmost part of Figure 5.3) are trained over these images, and the threshold-driven ensemble approach takes place for deciding which action to perform the day after the observations, as described in what follows. As mentioned before, this work as been published in [131]

5.1.1 Trading strategy

We design our system to simulate a classic *intraday trading* strategy. This strategy generally consists of buying or selling a specific financial instrument (in our case, the S&P500 index *future*), by making sure that any open position is closed before the market closes in the same trading day. Specifically, we model our strategy such that, for each single trading day, the final output of our system is one of the following actions:

- a **long** action, which consists of buying the stock, and then selling it before the market closes;
- a **short** action, which consists of selling the stock (using the mechanism of the *uncovered sale*), and then buying it before the market closes;
- a **flat** action, which consists of deciding not to invest in that day.

The ideal target of this strategy requires the system to choose the action that maximizes the economic return (*i.e.*, the *profit*) of the day, given a prediction about the stock price trend in that day (*i.e.*, whether the price will rise or fall). Thus, a *long* action is performed whenever our system predicts that the price will rise in that day; conversely, a *short* action is chosen whenever our system predicts that the price will fall in that day; last case, a *flat* action is performed whenever the system is not enough confident about the market behaviour.

5.1.2 Gramian Angular Fields imaging

The Gramian Angular Field (GAF) imaging is an elegant way to encode time series as images. This has been proposed by Wang et Oates in [132]. The main reasons which led to the definition of this approach regards the possibility to use existing pre-trained models, rather than training Recurrent Neural Networks from scratch or using 1D-CNN models. The last two models may result inconvenient.

In order to build the GAF images, first a rescaling of the real observations of the time series is needed; therefore, let $X = \{x_1, x_2, \ldots, x_n\}$ be the considered time series with n components, the rescaling to the interval [-1, 1] is achieved by applying the mean normalisation:

$$\tilde{x}^i = \frac{(x_i - max(X)) + (x_i - min(X))}{max(X) - min(X)} \tag{5.1}$$

Hence, the scaled series is represented by $\tilde{X} = \{\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_n\}$. This is transformed to a polar coordinates system by computing the angular cosine of the single components of the scaled time series:

$$\begin{bmatrix} \theta_i = arccos(\tilde{x}_i), & \tilde{x}_i \in \tilde{X} \\ r_i = \frac{i}{N}, & \text{with } t_i \leq N \end{bmatrix} \tag{5.2}$$

Finally, Gramian Summation Angular Field (GASF) and Gramian Difference Angular Field can be easily obtained by computing the sum/difference between the points of the time series:

$$\begin{aligned} GASF = [cos(\theta_i + \theta_j)] = \tilde{X}' \cdot \tilde{X} - \sqrt{I - \tilde{X}^2}' \cdot \sqrt{I - \tilde{X}^2} \\ GADF = [sin(\theta_i + \theta_j)] = \sqrt{I - \tilde{X}^2}' \cdot \tilde{X} - \tilde{X}' \cdot \sqrt{I - \tilde{X}^2} \end{aligned} \tag{5.3}$$

Figure 5.1 shows the process for transforming a time series to the GADF and GASF images. It is worth to notice that the equations in (6.3) produce a $1D$ matrix as an output of the encoding process. This matrix actually represents a heatmap, whose values range from 0 (*blue*) to 1 (*red*). In a successive step, we applied the RGB color map to the image, thus resulting in a three channel matrix (further details on this process can be found in [132] and [133]). Note that the application of the color map is not strictly required by our approach; however, preliminary experiments showed us that applying the color map to the images led us to obtain better results and, additionally, to achieve a faster network convergence and stability.

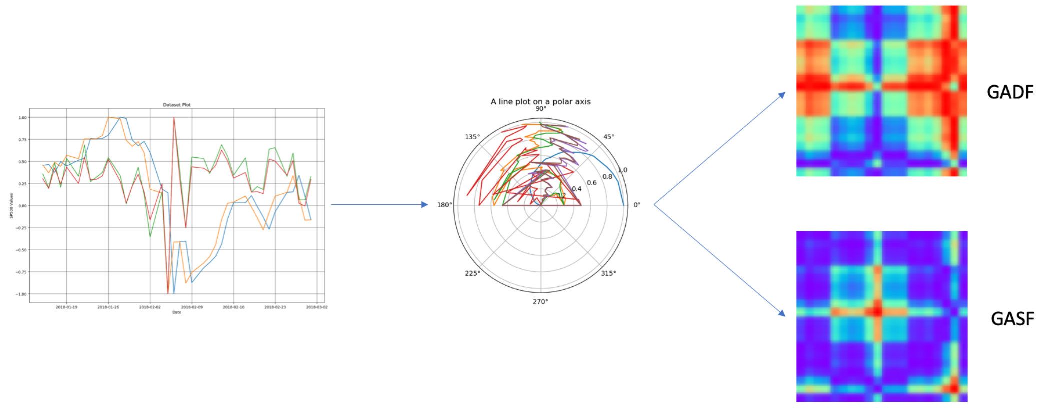


Figure 5.1: The figure shows the process leading to the generation of the Gramian Angular Fields images: from left to right, the data are first plotted and then the coordinate system is transformed to a polar plane; finally, the GADF and GASF images are generated according to the function defined in Equation 6.3.

5.1.3 Multi-Resolution Time Series Imaging

The time-series data show a quite important factor, that is the variation of a feature across the time and, therefore, how quickly data change. The speed which regulates the change of the features provides many insights about the evolution of the event: unfortunately, this peculiarity is hidden or even impossible to identify when data granularity is too coarse. As a trivial example, let us think to how important is the granularity of information in weather forecasting: an actual changing from sunny to rainy which occurs in few minutes gives different information with respect to the same change in 24 hours period. Moreover, often the observations contained in a time series are not done at the same interval of time, i.e. the distance between two consecutive observations is not always the same, thus forcing the researcher to aggregate the data for uniforming this distance through the entire time series. Given the two factors described above, our approach proposes a multi-resolution imaging, which aggregates the data under K different intervals of time, thus creating K different, but analogous, time series. Let $D = \{T, F_1, F_2, \ldots, F_N\}$ be the original time series in which T defines the moment in which the observations of a given event are done, and F_i with $i = 1, 2, \ldots, N$ are the features describing the event. Given two consecutive observations in D, let us say D_j and D_{j+1}, let I_D be the distance between them in terms of time. The new K aggregated time series are $D_1, D_2, \ldots, D_K$, and the interval between two consecutive observations is $I_{D_k} > I_D$ for each $k = 1, \ldots, K$.

Figure 5.2 shows the composition approach in which (a), (b) (c) and (d) are four GADF images built from four time-series which differ for their aggregation intervals. The composition aims at building a unique image, which considers the evolution of the time series in a fixed period of time.

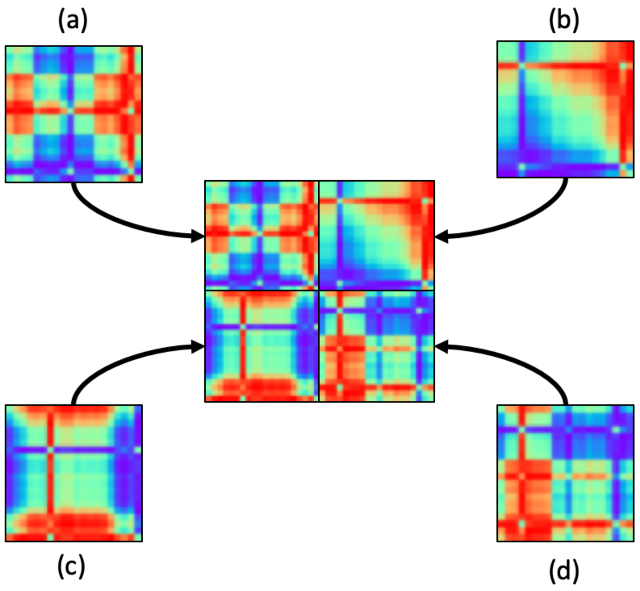


Figure 5.2: The policy of composition of the multi-resolution GADF images is shown above; in particular, (a), (b), (c) and (d) refer to the same label, but the observations considered in each are aggregated in four different ways.

5.1.4 Ensemble of CNNs

Once the time series are converted to GADF images, the training phase can take place. Figure 5.3 shows the architecture of the Convolutional Neural Network involved (on the left). It consists of a simplified version of the VGG-16 network [134], composed by 5 convolutional layers and a fully-connected one. Although our approach is independent of the network adopted, our choice was motivated by the fact that very deep networks were not suitable for the task, given the low number of samples at our disposal (SP500 only has a few thousand of daily samples). Indeed, our early tests showed that the simplified VGG led us to significantly improve the results, learning stability and execution times, both when compared to the standard VGG-16 and ResNet [135] architectures.

Additionally, note that no padding is added, therefore the convolutional layers produce an output of the same size of the input. The activation function in each of them is a ReLU, while in the classification layer the softmax function is used.

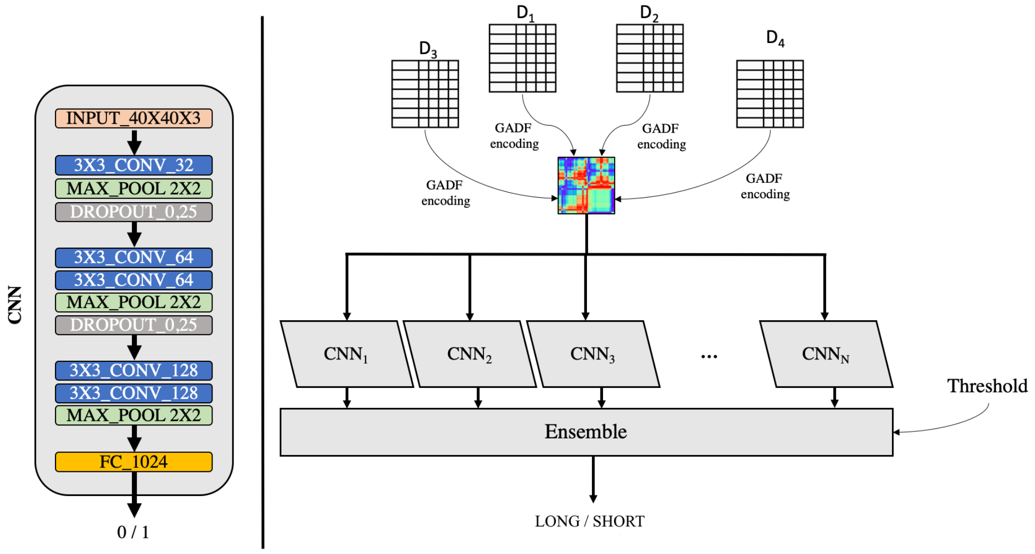


Figure 5.3: On the leftmost side of the figure the architecture of the Convolutional Neural Network is shown. On the rightmost side, the overall process of the proposed trading system is depicted.

The ensemble configuration involves the training of twenty Convolutional Neural Networks, each with a different weight initialization method; the configuration parameters for each network are detailed in Table 5.1. The choice of defining each network with different weight initialization methods arises from the need of excluding most of the randomness factors which can affect the final prediction. This ensures that the answer of the ensemble originates from a proper convergence of the weights, no matters how they are initialized. Once the neural networks are trained, the test samples are given as input to all the networks. The final decision of the ensemble is taken by applying a majority voting based approach, which returns a specific output $r \in \{0, 1\}$, according to the percentage of networks which agree with the same classification. In the experimental results section, six different thresholds (related to the agreement percentage of the networks) are tested, from 0.5 up to 1, according to the percentage of networks which agree to the same classification.

Table 5.1: In the table, for each initialization method, the configuration settings are reported. The "*seed*" column indicates the seed number applied for generating the random weights, according to the used initialization function. The first 10 networks have the "*seed*" set to "*None*", meaning that randomised numbers differ in each execution. The other 10 networks have a fixed "*seed*" meaning that randomised numbers are generated "*seed*". For all configurations, Adam has been used as an optimiser.

Init. method	**Configuration parameters**	**seed**
Orthogonal	gain=1.0	-
lecun_uniform	-	-
VarianceScaling	scale=1, mode='fan_in', distr.='normal'	-
RandomNormal	mean=0.0, stddev=0.05	-
RandomUniform	minval=-0.05, maxval=0.05	-
TruncatedNormal	mean=0.0, stddev=0.05	-
glorot_normal	-	-
glorot_uniform	-	-
he_normal	-	-
he_uniform	-	-
Orthogonal	gain=1.0	42
lecun_uniform	-	42
VarianceScaling	scale=1, mode='fan_in', distr.='normal'	42
RandomNormal	mean=0.0, stddev=0.05	42
RandomUniform	minval=-0.05, maxval=0.05	42
TruncatedNormal	mean=0.0, stddev=0.05	42
glorot_normal	-	42
glorot_uniform	-	42
he_normal	-	42
he_uniform	-	42

5.2 Experimental Settings

This section describes the settings of the experimental phase. In particular, first, in Subsection 5.2.1 the dataset S&P500 is presented and described along with the Buy-and-Hold investment strategy applied on it. This represents the benchmark comparison method for the majority of the trading systems. Then, in Subsection 5.2.2, the validation approach is defined. Finally, in Subsection 5.2.3 the ensemble policy is shown.

5.2.1 Evaluation and S&P500

The S&P500 (Standard & Poor's 500) is maybe the most important U.S. index. Born in 1789, at the beginning it only consisted of 90 titles. Since 1957, the year corresponding to when computers started to be actively applied to the financial market, the number of quoted companies has grown up to 500. It followed that S&P500 became one of the most influencing market in the U.S., even overcoming the Dow-Jones index. For the trading operations the S&P500 Futures have been used. They consist of futures contracts on a stock or a financial index. These derivative securities are used by investors and portfolio managers to hedge their equity positions against a loss in stocks; in other words, S&P500 Index is used by those who want to hedge risk over a certain period of time.

Currently, S&P500 index is one of the most widely traded index future contracts in the United States and it is computed by multiplying the S&P500 value by 50 USD. As an example, if the S&P500 is at a level of 2,500, then the market value of a future contract is 2,500 $\times$ 50 USD or 125,000 USD.

The S&P500 data are publicly available on many platforms over Internet; also, these can be downloaded at different levels of granularity, according to the scope of the research that is being carried out. The S&P500 dataset downloaded from Trading View Software has been used for the experiments. Historically the data are tracked and gathered once every fifteen seconds, but some of the platforms only provide data daily. More in detail, for each of the acquired records, the following data are available:

- Date of the observation: *mm/dd/yyyy*;
- Interval of the observation: *hh:mm:ss*;
- Open: the value the index has opened in the specified date;
- Close: the value the index has closed in the specified date;
- High: the highest value reached by the index in the specified date;
- Low: the lowest value reached by the index in the specified date.

We decided to use S&P500 as a benchmark for several reasons. First of all, because it is one of the most relevant stock markets in the world. The second reason is that it is extremely challenging to forecast its behaviour. Indeed, it shows a high level of volatility, where this measure indicates the mean fluctuation of the market over a certain time interval. The practical importance of this factor relies in the assumption that there is a tight correlation between the mean fluctuation and the quantification of the risks related to the asset [136]. In the recent years, many studies have been conducted in this area to the point that a specific branch of the market-related research focuses on the prediction of this fluctuation [137]. On top of this, we can use the performances of the Buy-and-Hold strategy (hereafter shortened as B&H) applied on S&P500 as a direct benchmark. B&H is both a strategy easy to replicate, and also an extremely significant competitor when considering a time frame where the market performs a large and quite constant growth. Indeed, B&H is a passive investment strategy where an investor buys stocks and holds them for a long time, with the hope that stocks will gradually increase in value over a long period of time. This strategy works as follows: given an investment period, B&H "*buys*" a stock at the beginning, "*holds*" it for the entire investment period, and *sells* it at the end. The net profit is the difference between the price at the end period and the price at the beginning. In Figure 5.4, the selected time frame for comparing our approach against the B&H strategy is shown (from 2009-02-01 to 2014-07-31); it is easy to notice that this investment period is extremely favourable for B&H and extremely challenging for our approach.

5.2.2 Walks' Definition

With respect to the common cross-validation approaches, which are typically applied when dealing with image classification, like the Leave-One-Out cross validation (LOOCV) or the k-fold cross validation, time series data need a more specific purpose approach, since they need to consider the semantic linking between the observation at time t and the one at time $t+1$. The walk-forward validation strategy properly fits in this scenario since the folds which are considered for the validation are temporally split, and internally processed as one sample. As a consequence, the use of GADF helps maintaining the correlation between two consecutive observations, thus keeping unaltered the semantic of the succession.

For our own research we considered an investment period which goes from February 1st 2000 to January 30th 2015, for a total of 16 years and 4569 observations which are related to the actual days in which the financial market has been opened. Each observation has been labelled according to the difference $Close - Open$ of the day after which is: F

- 1, if the close-open value of the day after is positive;
- 0, otherwise

The data are divided in training, validation and testing sets; in Table 5.2, the walks are shown: in particular, each model has been trained over a period of

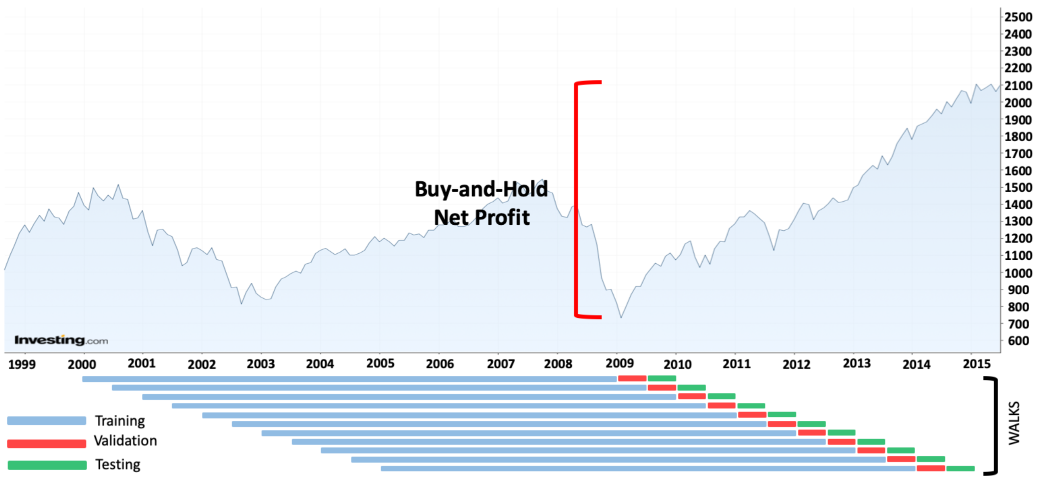


Figure 5.4: The net profit obtained by the application of the Buy-and-Hold strategy is shown over the period which has been taken into account for the experiments (https://www.investing.com/indices/us-spx-500-futures).

ten years, validated over the following six months and finally tested over the last six months.

Table 5.2: The eleven walks used for training, validation and testing the models and how they are composed.

	training	**validation**	**test**
#	**since → to**	**since → to**	**since → to**
1	2000-02-01 → 2009-01-30	2009-02-01 → 2009-07-31	2009-08-02 → 2010-01-31
2	2000-08-01 → 2009-07-31	2009-08-02 → 2010-01-31	2010-02-01 → 2010-07-30
3	2001-02-01 → 2010-01-31	2010-02-01 → 2010-07-30	2010-08-01 → 2011-01-31
4	2001-08-01 → 2010-07-30	2010-08-01 → 2011-01-31	2011-02-01 → 2011-07-31
5	2002-02-01 → 2011-01-31	2011-02-01 → 2011-07-31	2011-08-01 → 2012-01-31
6	2002-08-01 → 2011-07-31	2011-08-01 → 2012-01-31	2012-02-01 → 2012-07-31
7	2003-02-02 → 2012-01-31	2012-02-01 → 2012-07-31	2012-08-01 → 2013-01-31
8	2003-08-01 → 2012-07-31	2012-08-01 → 2013-01-31	2013-02-01 → 2013-07-31
9	2004-08-01 → 2013-07-31	2013-08-01 → 2014-01-31	2014-02-02 → 2014-07-31
10	2004-08-01 → 2013-07-31	2013-08-01 → 2014-01-31	2014-02-02 → 2014-07-31
11	2005-02-01 → 2014-01-31	2014-02-02 → 2014-07-31	2014-08-01 → 2015-01-30

5.2.3 CNN training and Ensemble policy

According to the considerations exposed in Section 5.1.3, we used $K = 4$ for the evaluation of our approach. Starting from the original time series of S&P500, in which the observations are sampled at intervals of 5 minutes, the data have been aggregated according to f4 new intervals:

- 1 hour
- 4 hours
- 8 hours
- 1 day

This means that starting from the original time series, we aggregated the data in four different ways and from each we selected 20 samples for predicting the market-day after (20 1-hour blocks make one GADF for the first time series, 20 4-hours blocks make one more GADF image, etc).

The $Close - Open$ value has been computed for each sample, and a $20X20X3$ GADF image has been built and composed according to the process shown in Figure 5.2. By following this procedure, the aggregated GADF image will have dimension $40X40X3$. For each of the walks defined in the previous section, the ensemble of CNNs is executed over the test samples; the majority voting approach works as follows: each of the CNN gives as output a single-value which is 0 or 1 whether the CNN suggests to perform a short or a long action, respectively. When working in ensemble, the final prediction is taken according to the answer of all the CNNs, according to the trading strategy defined in 5.1.1. Note that in cases where not all the CNNs agree with the same result, a coverage-base approach is used: let N be the number of networks involved in the ensemble, $t \in [0.5, 1]$ be a threshold which indicates the required percentage of agreement, and `A` be the most predicted action (between long or short) by the nets for the considered day. Then, the system performs `A` in that day if and only if at least $n > tN$ networks have voted `A`; conversely, if the agreement threshold is not reached, the system does not perform any operation and just holds.

5.3 Results and Discussion

In this section, the quantitative and qualitative results are shown and discussed. The results of the proposed approach have been compared against the following benchmarks:

- **B&H Strategy**: as explained in Section 5.2, this represents the benchmark comparison method for the majority of the trading system approaches;

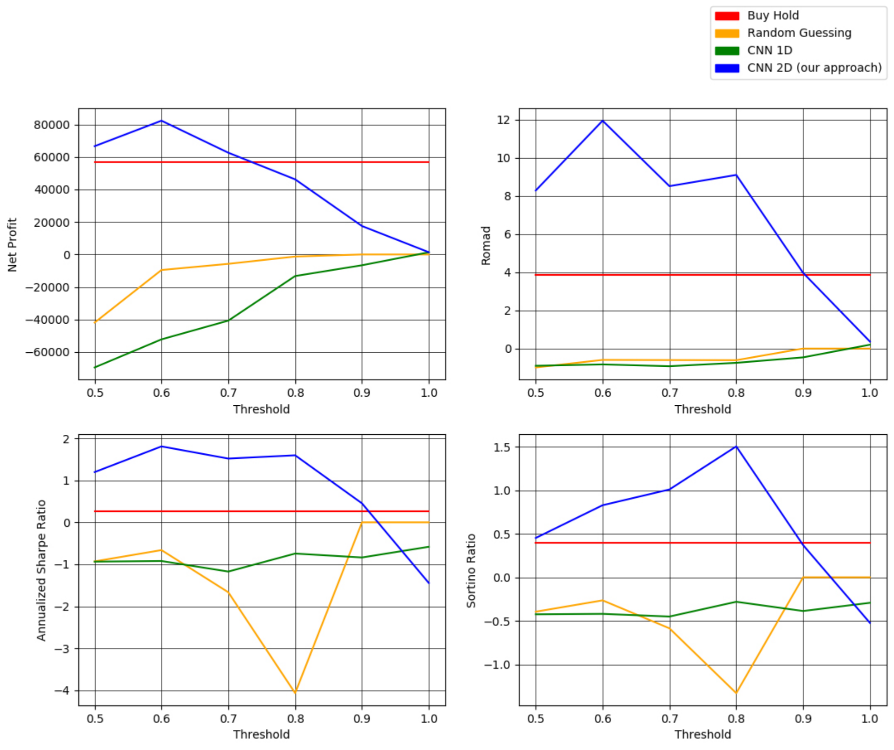


Figure 5.5: The figure shows the comparison among the tested approaches. In particular, the blue, orange and green lines represent, respectively, the results obtained by applying our approach, a random guessing approach and a 1D-CNN. The red line evidences the results which are obtained by the Buy-and-Hold strategy. In each plot, the x-axis indicates the ensemble threshold considered, while in the y-axis it is shown the obtained Net Profit, the Romad value, the Annualized Sharpe Ratio and the Sortino Ratio, respectively, from the upper-leftmost image to the bottom-rightmost one.

- **Random Guessing**: this easy technique exploits 10 random classifiers, included in an ensemble that using a majority voting approach tries to perform long or short operations. The comparison against this random predictor serves as proof that our approach does not act randomly, that the performed actions (longs or shorts) have a strong basis, and that the criteria are correctly learnt from the past trend of the market;

- **1D-CNN**: this approach does not apply the GADF transformation to the time series; therefore, the time series are directly applied and processed by 1-dimensional Convolutional Neural Network. This test aims at showing the benefits of the GADF transformations in the proposed approach.

Figure 5.5 shows the performances of the simple trading system based on our forecasting approach (in blue) against those described before: the B&H strategy (in red), the 1D-CNN (in green) and the Random Guessing (in orange). The performances are computed over all the walks that have been taken into account; on the x-axis of of each plot the considered threshold for the ensemble is indicated. Overall, the thresholds from 0.5 to 0.7 let us obtain the best achievements. As shown in the plot, the proposed approach is the only one capable of sensitively overcome the benchmark performance of the B&H. However, a peak in the performance is appreciable with the threshold set to 0.6, in which the highest net profit is obtained. The quantitative results are shown in Table 5.3 and further confirm the good qualitative scores. Moreover, the table shows the big difference between the net profit obtained by our approach and the B&H strategy. Using ensemble thresholds higher than 0.7, the results tend to degrade, mainly due to the fact that it becomes harder to obtain a total agreement among the nets in the ensemble. Anyway, even with a threshold set to 1 (meaning that all the nets must agree to trigger a long or short action), we managed to earn 1.375 dollars; thus, we still do not lose money, even though the risk is quite high, as shown in the Sharpe Ratio and Sortino Ratio plots. In general, the good performances for all the threshold values are a clear indicator of the robustness of our proposed approach.

In addition to comparing our results to the benchmark, we have also considered the approach proposed by Calvi et al. [5], which consists of the daily prediction of the S&P500 index, as in our case, but by using a *Support Tensor Machine* (STM) as a predictor (defined as a tensor extension of the better known Support Vector Machine). Figure 5.6 shows the gap between the net profit given by our method, and those returned by the Buy&Hold and the work in [5]. The comparison does not take into account the market period related to the 2008's Financial Crisis (whose effects impacted the years between 2007 and early 2009), when it results unfair to overcome the B&H strategy, due to the dramatic performances of the index in this period.

Table 5.3: The table shows the quantitative results of the proposed approach. For each metric, the best result is highlighted in bold.

	Threshold	**Coverage (%)**	**Accuracy (%)**	**Net Profit (USD)**	**Romad**	**ASR**	**Sortino Ratio**
Proposed Approach	0,5	**95,433**	52,638	66.625	8,289	1,196	0,452
	0,6	72,423	54,810	**82.312**	**11,95**	**1,808**	0,828
	0,7	47,716	56,564	62.600	8,517	1,518	1,009
	0,8	22,950	**56,632**	46.212,5	9,105	1,596	**1,504**
	0,9	7,494	53,125	17.487,5	3,97	0,452	0,370
	1,0	1,990	55,882	1.375	0,365	-1,445	-0,524
1D-CNN	0,5	94,906	47,254	-69.525	-0,904	-0,936	-0,425
	0,6	74,824	46,791	-52.250	-0,832	-0,921	-0,420
	0,7	53,688	46,564	-40.725	-0,927	-1,172	-0,451
	0,8	33,723	46,875	-13.287,5	-0,749	-0,743	-0,281
	0,9	15,456	48,106	-6.650	-0,463	-0,836	-0,387
	1,0	7,494	49,218	1.337,5	0,2	-0,582	-0,293
Random Guessing	0,5	82,552	48,297	-41.912	-0,991	-0,930	-0,395
	0,6	26,639	50,109	-9.575	-0,596	-0,661	-0,265
	0,7	4,449	42,105	-5.775	-0,604	-1,665	-0,586
	0,8	0,292	20,0	-1.275	-0,610	-4,067	-1,331
	0,9	0	0	n.a.	n.a.	n.a	n.a
	1,0	0	0	n.a.	n.a.	n.a.	n.a.
B&H	-	-	-	56.600	3,847	0,258	0,394

Conversely, the considered period (from late 2009 until 2016) highlights a strong increase of the S&P500 index, thus improving the Buy&Hold performances. Nevertheless, our method is able to outperform such performances, which is not the case of the approach in [5].

Finally, note that, in the Financial Forecasting domain, the prediction accuracy is very close to 50%, mainly due to the high variability of the indexes which complicates a lot the decision between long and short actions. In this scenario, using a single CNN (both in terms of prediction and, where appropriate, in terms of probability of the prediction) usually introduces a major drawback: the initialization seed may significantly affect the final result. Moreover, focusing on improving the accuracy does not usually imply the net profit to grow accordingly. It follows that, by slightly changing the initialization, the same network could give worse results for the same period, or in any case could behave too differently when tested on different markets. This reveals the need of stabilising the results; the ensemble architecture, together with different weight initialisation policies, has shown to have a much more robust behaviour, allowing us to obtain more stable results and thus alleviate the randomness. As a result, according to the set thresholds, we perform a *long* operation only when the ensemble suggests to buy when a certain ensemble voting policy is satisfied; otherwise we perform a *short* operation. The mitigation of the randomness yields two simple but significant consequences:

- when we lose, we tend to lose very little;
- when we win, we tend to win considerably.

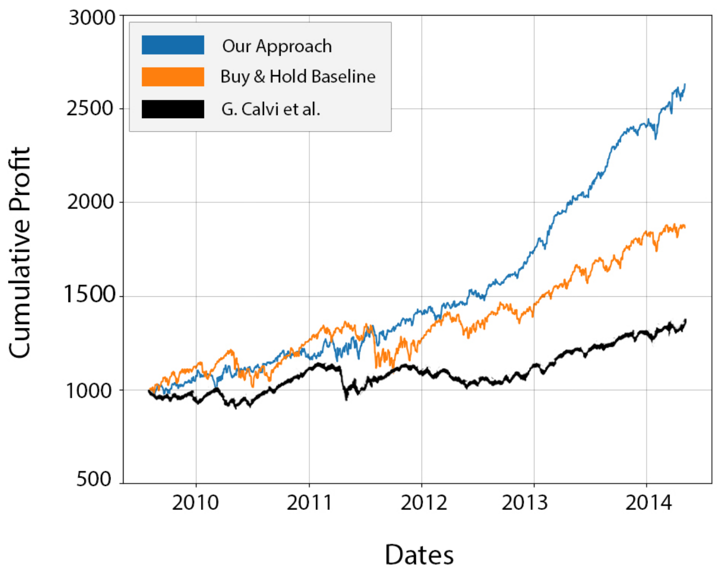


Figure 5.6: The above plot shows the comparison, in terms of cumulative profit, between the proposed approach (in blue) and the approach in [5] (in black). Finally, in orange, we show the Buy&Hold benchmark as a benchmark.

This result is to be considered particularly significant, thanks to the capability of our approach of beating the B&H strategy in the years in which the latter performs well.

Chapter 6

A Multi-Layer and Multi-Ensemble Stock Trader Using Deep Learning and Deep Reinforcement Learning

6.1 Proposed Approach

As it can be noticed from the literature on RL research, there are a few explorations on maximizing ensembling steps on the reward-based training pipeline, with most of the works in this regard relying solely on one individual agent that receives one or multiple sources of information as input, and the output is done by one single agent that is trained on a customized number of iterations.

In our work, we propose to do exactly the opposite by maximizing ensembling steps in a three layer stock trader. Our approach receives stacked inputs from other classifiers and fuses its set of classifications through several training iterations, integrating these steps on an RL trader agent. Our three layer-stock trader, whose pipeline is better illustrated in Figure 6.1, is composed of the following layers:

1. Layer #1: **Stacking/pre-processing layer:** this layer acts on stock market data at different time resolutions, converting these time series data to images and using intra-day trading signals generated by 1000 CNNs as meta-features to be used by the next layer.

2. Layer #2: **Reinforcement Meta Learner:** we consider a reward based classifier to process the outputs from the previous layer, an ensemble process that is usually called stacking [138]. The stacking is

performed by a meta-learner, which is, in our case, based on Deep Double Q-Learning [139].

3. Layer #3: **Ensembling Layer:** this layer fuses different signals of different training iterations of the meta learner in order to get a final decision.

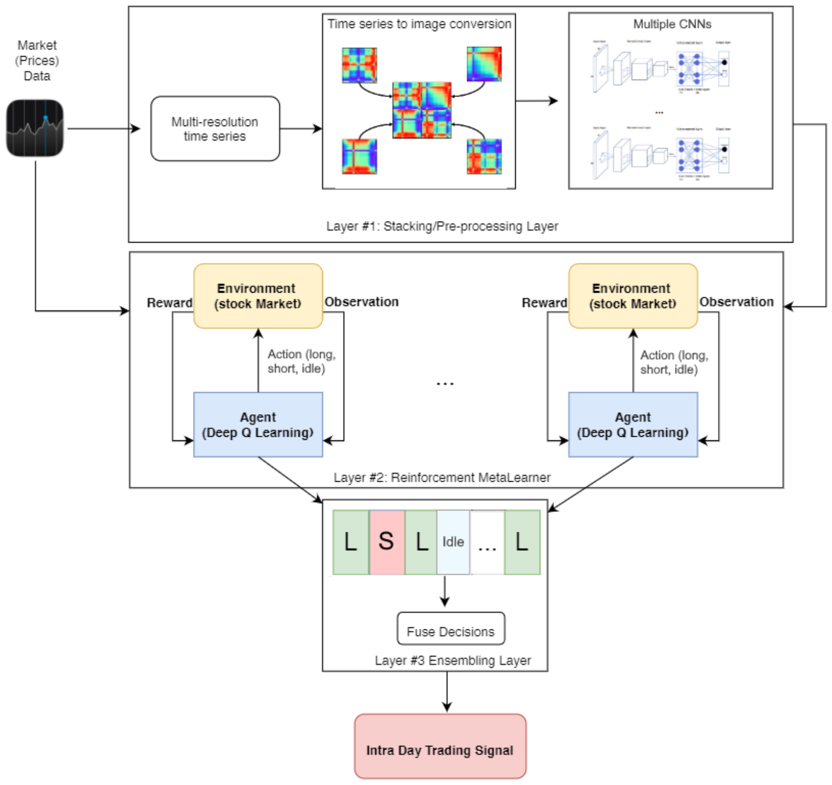

Market (Prices) Data
Multi-resolution time series
Time series to image conversion
Multiple CNNs
Layer #1: Stacking/Pre-processing Layer
Reward
Environment (stock Market)
Observation
Action (long, short, idle)
Agent (Deep Q Learning)
...
Layer #2: Reinforcement MetaLearner
L
S
L
Idle
...
L
Fuse Decisions
Layer #3 Ensembling Layer
Intra Day Trading Signal

Figure 6.1: The proposed three layered multi-ensemble approach. The first layer stacks decisions from CNNs, which are then used as observations (or states) in a reward-based meta-learner (second layer) in different training iterations. Finally, a last layer fuses the final decisions in a final trading signal.

We discuss the details of each layer in the next subsections.

6.1.1 Layer #1: Stacking trading signals with Convolutional Neural Networks

Several approaches in the literature have used CNNs to perform stock trading decision [140, 141, 142], treating the stock trading problem as a Computer

Vision application by using time-series representative images as input. Following this approach, CNNs can highlight in their deeper layers details of these images that are difficult to find in their numerical format, helping in performing accurate classification and, therefore, better trading.

In the first layer of our trading algorithm, we leverage the benefits of these networks by generating a series of meta-features using hundreds of CNNs. This approach, inspired by the work of Barra *et al.* [140], is based on three steps: (i) time-series-to-image conversion; (ii) different CNNs processing; and (iii) fusing of trading results.

For converting time series to images, the authors consider the Gramian Angular Field (GAF) imaging inspired from Wang and Oates [108]. To build such images, re-scaling of time-series are needed. Let $X = \{x_1, x_2, \dots, x_n\}$ be a time series with n components, the following normalization, usually called the min-max normalization [143] is done as follows:

$$\tilde{x}^i = \frac{(x_i - max(X)) + (x_i - min(X))}{max(X) - min(X)} \tag{6.1}$$

The scaled series, represented by $\tilde{X} = \{\tilde{x}_1, \tilde{x}_2, \dots, \tilde{x}_n\}$ are then transformed to polar coordinates system by the following equation:

$$\begin{bmatrix} \theta_i = arccos(\tilde{x}_i), & \tilde{x}_i \in \tilde{X} \\ r_i = \frac{i}{N}, & \text{with } t_i \leq N \end{bmatrix} \tag{6.2}$$

Finally, Gramian Summation Angular Field (GSAF) and Gramian Difference Angular Field (GDAF) matrices can be easily obtained by computing the sum/difference between the points of the time series:

$$\begin{aligned} GSAF(i,j) &= [cos(\theta_i + \theta_j)] = \tilde{X}' \cdot \tilde{X} - \sqrt{I - \tilde{X}^2}' \cdot \sqrt{I - \tilde{X}^2} \\ GDAF(i,j) &= [sin(\theta_i + \theta_j)] = \sqrt{I - \tilde{X}^2}' \cdot \tilde{X} - \tilde{X}' \cdot \sqrt{I - \tilde{X}^2} \end{aligned} \tag{6.3}$$

The result of such transformations are 1-channel 2D matrices that represent a heatmap, whose values range from 0 (*black*) to 1 (*red*). In a second step, a colormap is applied to the input matrix, resulting in a three channel output matrix. In the work of Barra *et al.* [140], best results were reported using GDAF.

Converting stock market time series to images in this approach has the limitation of not generating too big images, making such inputs unfeasible to Deep Convolutional Neural Networks. Time-series data show an important feature that is the variation of data in the time. They way the data change can, therefore, provide important interpretation about how an event evolves. Thus, the authors propose to aggregate the data under K different intervals of time, using these K time series as sub-regions of the final converted image.

This image has a final resolution of 40 × 40 pixels, and is processed by an ensemble of VGG-based CNNs, with the architecture better detailed in Figure 6.2. The final trading signal is the majority voting of the outputs.

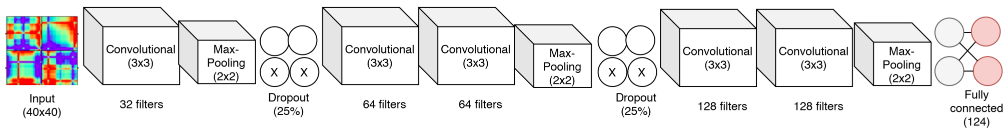


Figure 6.2: The VGG-based CNN used to pre-process stock market time series to image data.

Our extensions of the work in [140] are two-fold: (i) in [140], twenty different CNNs (each trained with a different weight initialization) are fused to provide the final (daily) decisions. We noticed that the adoption of a naive consensus mechanism to fuse these CNN outputs did not allow the approach to be sufficiently robust in markets with different characteristics. Hence, in this new work, we improved the method by exploiting Reinforcement Learning to find the optimal policy to fuse these CNN outputs, with the aim of favouring the most reliable networks, and penalising the less precise ones; (ii) to do so, we first extend the numbers of CNN used, by considering 10 best training epochs over 100 CNNs, with different initialization parameters (not previously used in [140]), to generate daily trading decisions. Then, these 1000 array of decisions are used, in a novel way, to generate meta-features to be employed as inputs of several RL agents. Therefore, the 100 CNNs act as a first layer (or feature generation/pre-processing) of our new approach, not used to trade as the 20 CNNs from [140] are used for; and (iii), finally, we train these RL agents (with different experience of the environment) based on Neural Networks (Deep Q-Learning) with these features to automatically find the possible best ensembling policies, to perform a majority voting over these agents (as described in next Sections 6.1.2 and 6.1.3), in order to get final decisions to be tested in real-world trading scenarios.

6.1.2 Layer #2: Reinforcement MetaLearner

Reinforcement Learning (RL) [144] is a self-taught learning process which can be represented by a Markov Decision Process (MDP) [145]. An MDP is a 4-tuple (S, A, P_a, R_a) where:

- S is a set of states;
- A is a set of actions (in the specific case of *reinforcement learning* this is the set of actions that the agent is allowed to do, thus in our case this set is finite);

- $P_a(s, s')$ is the probability that an action a in state s will lead to the state s';
- $R_a(s, s')$ is the immediate expected reward received in the transition from a state s to a state s' doing an action a.

The main goal of the Reinforcement Learning process is, therefore, to find a policy π that returns the highest reward. To perform such a task, the agent can try different policies in a training stage, but only one of those has the highest reward possible, which is called the *optimal policy*. Several approaches can be used to train RL agents, among them, a promising strategy is using evolutionary algorithms [146].

When applied in Machine Learning problems, the following analogies are usually made: (i) the state is a feature vector which represents the input of the agent (*e.g.*, in our case, a vector containing input data); (ii) the actions are the outputs of the Machine Learning agent (*e.g.*, the trading decision to do); and (iii) the reward function is built at a training stage and returns higher rewards to correct actions (*e.g.*, the market profit generated by the local trading decision). Thus, differently from supervised learning (when agents learn from annotations) and unsupervised learning (where agent searches for hidden structures in the data, to identify clusters), in Reinforcement Learning the agent learns how to maximize a reward function in a given environment.

In our proposed Reinforcement Trading (RT), the trading signals vector from the previous layer is described as $\{p_1, ..., p_n\}$, and the return at time t is defined as $r_t = p_t - p_{t-1}$. Trading decisions such as long, flat and short are represented as $A_t \in \{1, 0, -1\}$. Therefore, in RT, the profit R_t is calculated as [147]:

$$R_t = A_{t-1} r_t - c|A_t - A_{t-1}| \tag{6.4}$$

where c is the transaction cost paid to the brokerage company when two consecutive decisions are done, or $A_t \neq A_{t-1}$.

Hence, the objective of RT is maximizing the reward over a period T, or

$$\max_{\Theta} \sum_{t=1}^{T} R_t | \Theta, \tag{6.5}$$

where Θ is the family of parameters of the model (agent). For example, if the model is a Neural Network, then $\Theta = \{w, b\}$, where w are the weights and b are the biases that should be trained to maximize the objective.

In our approach, we use the benefits of Reinforcement Learning classifiers to act in a stacking scenario. Here, the RL agent acts like a *meta-learner*, using the outputs from previous layer as states. In its training, the agent learns to maximize rewards, which is calculated as:

$$Reward = \begin{cases} close - open & \text{if action is } long \\ -(close - open) & \text{if action is } short \\ 0 & \text{if action is } flat \end{cases} \tag{6.6}$$

where *open* and *close* are, respectively, the opening and closing prices of the mar

Our meta-learner in the considered day earner is based on Double Q-Learning (DQL) agents [148]. In such an algorithm for Deep Reinforcement Learning (DRL), Deep Q Networks (DQN) are used in order to learn a reward-inspired function called Q-function $Q(x, a)$, which is disposed in a table. This function is calculated as:

$$Q(x, a) = D(x, a) + \gamma \max_a Q(y, a), \tag{6.7}$$

where x and y are a states, a is an action, $D(x, a)$ is the reward from taking action a and γ is the discount factor, which controls the contribution of rewards in the future. Therefore, in Equation 6.7, the Q-value yielded from being at state x and performing action a is the reward $D(x, a)$ plus the highest Q-value possible from the next state y also performing action a, discounted by the discount factor.

The update rule for finding new Q-function approximations is based on the following criterion:

$$Q(x, a)_{new} = Q(x, a)_{old} + \alpha[(D(x, y, a) + \gamma \max_b Q(y, b) - Q(x, a)_{old})], \tag{6.8}$$

where α is the learning rate, $D(x, y, a)$ is the immediate reward from taking action a and moving from state x to y and $\max_b Q(y, b)$ is the maximum future reward considering all possible actions b in the new state y.

Such a parameterized table is learned through network weights in Deep Q-Learning, found through backpropagation of errors in a training stage. DQN loss functions are calculated as:

$$Loss = [Q(x, a, \Theta) - (D(x, a) + \gamma \max_a Q(y, a, \Theta')]^2 \tag{6.9}$$

Therefore, the goal of the network is to find parameters Θ that minimize Equation 6.9.

In a testing scenario, a DQN processes the input (state) and returns Q-values for each action to take. Then, the action with the highest Q-value will be chosen to be executed at that state. Therefore, for a state with n dimensions and an action space of m possible actions, the Neural Network is a function $R_n \rightarrow R_m$.

Some important features of our considered DQL are the target networks [148] and dueling networks [149]. The first one is used just to calculate the target from Equation 6.10 below. It is a network similar to the main network, and its weights w' are initially the same as the weights w of the main network. However, new weights are copied every t steps from the main network and remain the same in the other steps. When using target networks, the weights update of DQNs happens as it follows:

$$\triangle w = \alpha[(D(x, y, a) + \gamma \max_a Q(y, a, w)) - Q(x, a, w)] \bigtriangledown_w Q(x, a, w), \quad (6.10)$$

where $\triangle w$ is the change in weights to perform in the training stage, $Q(x, a, w)$ is the current predicted Q-value and $\bigtriangledown_w Q(x, a, w')$ is the gradient of the current predicted Q-value. The part of the equation $D(x, y, a)+\gamma \max_a Q(y, a, w)$ $-Q(x, a, w)$ is commonly known as Temporal Difference (TD) error. Finally, the part $D(x, y, a) + \gamma \max_a Q(y, a, w)$ is also called *target*.

The use of target networks is justified since the same parameters (weights) are used for estimating the target $D(x, y, a) + \gamma \max_a Q(y, a, w)$ and the predicted Q-value $Q(x, a, w)$ in Equation 6.10. As a consequence, there is a strong correlation between the target and the network weights w. This means that, at each training step, both the predicted Q-values and the target change. This causes an unstable training procedure, as the weights update goal will be finding a predicted Q-value that gets close to an always moving target. The use of a target network yields a better and fast training, and is composed of two steps: (i) the main DQN network helps selecting the best action (the one with the highest Q-value); and (ii) use the target network to calculate the target Q value of taking that action at the next state.

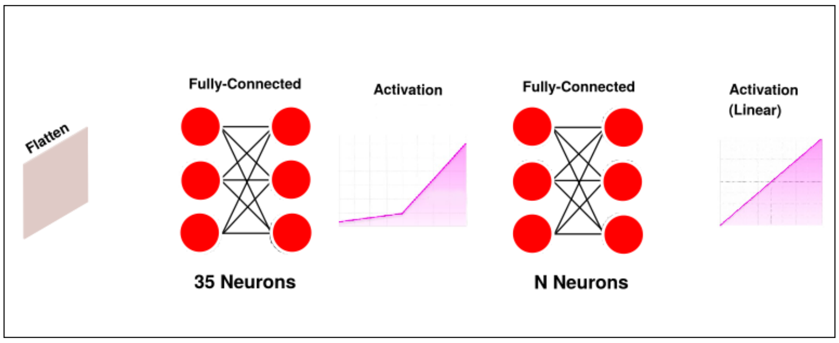


Figure 6.3: The Neural Network architecture of our proposed multi-DQN agents for stock trading. The activation of the first layer is chosen through optimization experiments.

Additionally, dueling networks [149] help the network to calculate each of the values $(D(x, y, a)$ and $\max_b Q(y, b))$ in Equation 6.7 separately. This is done through individual fully connected layers positioned after the penultimate layers of the DQNs, which estimate each of these values with a final aggregation layer that is then used to calculate $Q(x, a)$. To avoid backpropagation issues created by calculating Equation 6.7 in pieces, the aggregation layer includes subtracting the mean $\max_b Q(y, b)$ for all possible actions b. Therefore, this architecture helps accelerating the training, as the immediate reward $D(x, y, a)$ can be found without calculating the $\max_b Q(y, b)$ for each action at that state.

Figure 6.3 shows the architecture of the proposed main network used to stock trading, which includes the use of dueling and target networks in its final version. The main DQN is a very simple network composed of one flatten layer, one fully connected layer with 35 neurons. The activation of such neurons are activated after optimization experiments, which depend on the market considered (we discuss how it is done in Section 6.3.1 for one dataset), and N neurons fully connected layer with linear activation, where N is the number of decisions to take in the stock market (we chose $N = 3$, which are long, short and long operations).

6.1.3 Layer #3: Ensembling Multiple Learners

In this last layer, we propose the fusion of several multiple DQN agents trading signals, in order to take more decisions into account for trading. After multiple different training iterations (or epochs) of the above meta-learning agent with the environment, different actions in the market are done, as Figure 6.4 illustrates.

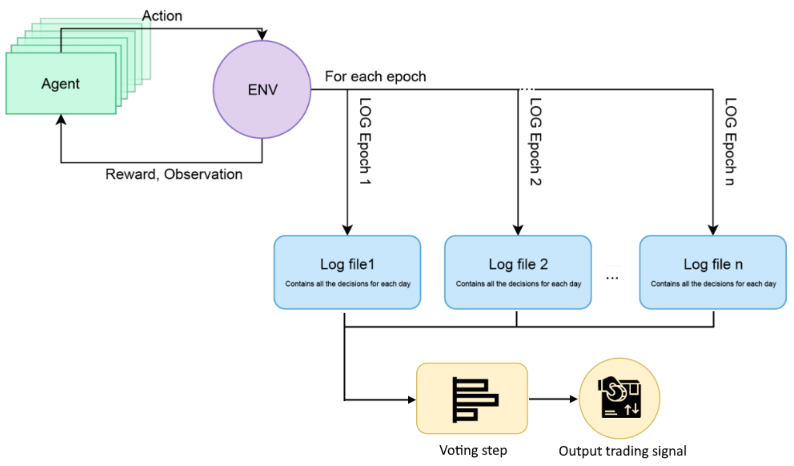


Figure 6.4: Fusion Layer pipeline of our proposed multi-layer agent. Multiple agents trained after multiple series of different iterations with the environment perform intra-day stock trading, done by choosing among different combinations of actions. The final action to take is decided by majority voting of decisions.

Given such outputs from the previous meta-learners, the final agent works in an ensemble (or *late fusion*) fashion, which considers the majority voting of decisions to generate the final trading signal. By taking a vector $L_t = \{l_1, l_2, ..., l_n\}$ of n iterations (epochs) decisions for a day t, where $l_i \in \{L, H, S\}$, the ensembling layer fuses them in just one trading signal l_t as the following:

$$l_t = Mode(L_t) \tag{6.11}$$

where $Mode(L_t)$ is the most frequent value in L_t. In our approach we consider L_t as a 25-dimensional vector (empirically chosen), which means that we ensemble 25 meta-learner training iterations decisions.

The idea behind this approach is that the same agent trained at different iterations can be complementary, as the agents have different experiences with the environment thus making the approach more robust against uncertainties of the stock markets behavior. Indeed, by ensembling the individual agents decisions through the majority voting, we require the final agent to perform a *long*, *flat* or a *short* action only when the fusion has a good confidence about the choice (*i.e.*, when there is a wide agreement among the individual agents).

6.2 Experimental Setup

This section describes the experimental setup adopted in this work. It reports the markets, metrics, state-of-the-art approaches considered for comparison and implementation details of the proposed approach.

6.2.1 Datasets

We perform the experimental validation of our approach by using historical data from the *Standard & Poor's 500* future index (S&P 500), as well as the *JP Morgan* (JPM) and *Microsoft* (MSFT) single stock indexes. These data can be usually found at different time resolutions, or *granularity* (*e.g.*, *5-minutes*, *10-minutes*, *1-hour*, *etc.*). The dataset structure is summarized in Table 6.1: each entry specifies the timestamp of the considered time slot (technically known as *candle*), among with its open, close, high and low prices. These prices are usually denoted by market points (1 point=50,00 USD for the S&P 500 future index, while 1 point=1,00 USD for single stocks). We run our first layer of CNNs over these datasets, thus extracting 1000 different predictions (long, short or flat) for each day of the chosen period, to be used as input for the second layer (the meta-learner).

Table 6.1: Dataset structure (1-hour time resolution); prices are expressed in market points.

Date	**Time**	**Open**	**High**	**Low**	**Close**
...	...	...	...	...	...
17/06/2011	19 : 00	7349	7363.5	7346	7351
17/06/2011	20 : 00	7352	7373	7350	7362
17/06/2011	21 : 00	7361	7376.5	7341.5	7374.5
...	...	...	...	...	...

6.2.2 Benchmarks

In order to make a complete comparison between the proposed approach, some other well-performing ensembling techniques, and literature benchmarks, we selected several significant benchmarks.

More in details, the benchmarks we consider can be divided into two groups. The first group includes some variations of our approach, where the second and third layers of our stack are replaced with alternative and well-performing non-RL ensembling techniques. They are:

1. the first layer only of our approach, through a majority voting over the 1000 CNN trading signals, which shows to be the strongest benchmark, with a very competitive behaviour in the considered periods;

2. the first layer only of our approach, but considering a threshold agreement of at least 50% of the same CNN trading signals;

3. the same first layer only of our approach, but now considering a threshold agreement of at least 70% of the same CNN trading signals.

On the other hand, the second group of benchmarks is represented by some literature works that propose trading techniques based on Machine Learning. As far as we know, however, the existing works that punctually report the used out-of-sample periods, as well as the results in terms of economic performance of their methods, are very few for intra-day stock trading. Among them, we have selected the following two:

1. the approach proposed by Sezer *et al.* in [150], called *CNN-TAr*, where the authors encode trading signals (*e.g*: EMA, SMA, *etc.*) into 15x15 images, and train a CNN to perform daily trading;

2. the technique presented by Calvi *et al.* in [151], where the authors propose a *Support Tensor Machine*, called *LS-STM*, *i.e.* a tensor extension of the best known support vector machine, to perform daily trading.

Finally, together with all the benchmarks mentioned above, for each period and market considered, we compare our proposed method against the Buy-and-Hold strategy, which consists of a passive investment where the investor buys the stocks at the first day of the considered period, and sells them at the end of the period. Since this strategy essentially replicates the behaviour of the market, it represents a valid benchmark to compare the performance of the proposed method against the market itself.

6.2.3 Implementation Details of the Proposed Approach

The proposed approach has been developed in *Python* using KERAS-RL library [152] and was run in an *Intel i7-8700k* with a *64-bit* Operating System (*Ubuntu Linux 18.04* for tge development and learning stages, *Microsoft Windows 10* for the simulation step) with *64 GBytes* of *RAM* and a Nvidia TITAN XP GPU. Our method includes several DQN agents, equipped with target networks and dueling architectures. In its second layer, it considers agents trained for 25 epochs/iterations (empirically chosen) and the final trading signal is given in the last layer of our method, by considering the majority voting of these decisions.

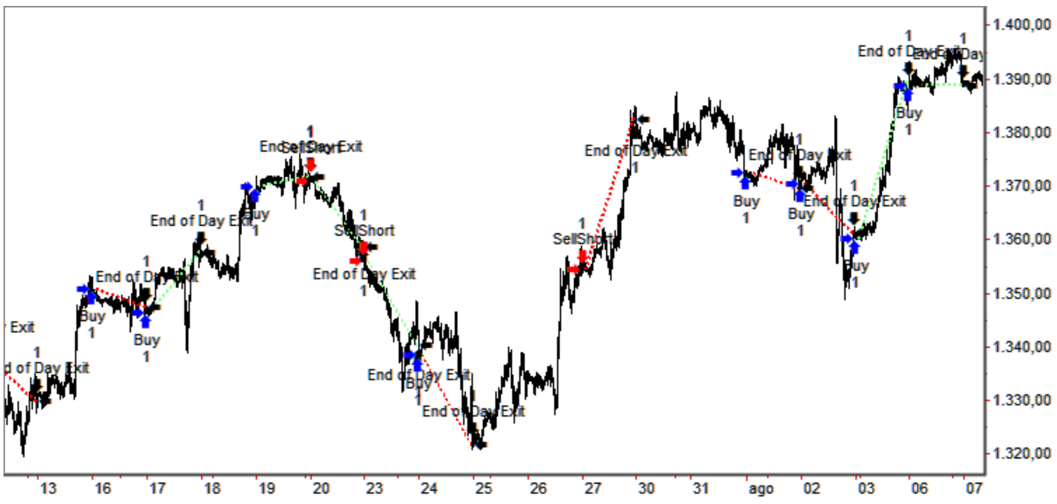


Figure 6.5: Example of real-world trading simulation through MultiCharts.

The output of our last layer is hence encoded into a *(datetime, decision)* `.csv` file. Then, each real-world trading simulation is performed through MultiCharts[1], a Windows-based trading platform. Figure 6.5 shows how MultiCharts implements our trading decisions: when performing *long* actions, the tool executes a buy order at the begin of the day, immediately after the market opens (recall we perform daily trading), and a sell order immediately before the market closes; for *shorts*, the opposite (i.e. it performs a sell order when the market opens, and a buy order when it closes, by exploiting the *uncovered sales* mechanism); finally, *flat* actions are simply ignored. Notably, the source code of our solution is available for download at a public repository[2].

[1] `https://www.multicharts.com/`
[2] `https://git.io/JRX4D`

6.3 Experiments

In this section, we validate the quality and trading performances of our approach through several simulations, in which we perform intra-day trading in several stock markets, through a *backtesting* mode. Our experiments consist of two stages: in the first stage, we perform a preliminary study on the metalearner parameters to optimize the second layer of our approach; then, we compare our proposed strategy against the benchmarks introduced in previous Section 6.2.2, in a real-world trading scenario.

6.3.1 Metalearner Parameters Optimization

Before showing the comparison with benchmarks and other relevant literature works, we report the outcomes of the preliminary parameter optimization performed to tune the second layer of our approach. To do so, we consider a small out-of-sample subset, that we call *validation set*, to analyze the results. To this, we exploit our S&P 500 dataset, spanning the date interval between February 1st of 2012 to September 5th of 2018 for training, and from September 6th of 2018 to March 5th of 2019 for validation.

The first set of parameters are the optimizers of the Deep-Q Networks. Indeed, this is a very important parameter, because the goal of the Reinforcement Learning network is to minimize the difference between the different Q-functions in Equation 6.9. Therefore, the optimizers update the weight parameters in Θ to perform such an optimization. The Loss Function acts as a guide to the optimizer, telling if it is moving in the right direction to the global minimum.

The second parameter comes from the RL dilemma known as *exploration versus exploitation*, that requires to choose between keep doing the normal learning process (exploitation), or try something new (exploration). In our deep-q agent, exploitation means always taking the action with the highest Q-value, and explorations means taking random actions with a given percentage of probability. We vary the probability of explorations in the greedy policy used in our metalearner.

Finally, the final parameter we want to tune is the activation of the 35-neurons layer of our Q-network. Activation functions are one of the most important components of a Neural Network model. They determine the output of a model, its performance and efficiency of training.

In summary, the possible values of each parameter are:

1. Adagrad, Adam, Adadelta, Adamax and Rmsprop, as optimizers;
2. 30%, 50%, 70% and 90%, as percentage of explorations;
3. TanH, Selu, Relu, Linear and Sigmoid, as activation functions.

Table 6.2 shows the top-10 results considering the Romad metric out of 100 experiments performed, showing their classification and trading performance.

Table 6.2: Top-10 results, in terms of Romad, of the parameter optimization experiment performed over the validation set, by considering different metalearner optimizers, layer activations and different levels of explorations in the metalearner policy. Classification metrics are percentages in [0, 1] range, while trading metrics are shown in market points (1 point = 50 USD).

Optimizer	**Explor.**	**Activator**	*Long Prec.*	*Short Prec.*	*Accuracy*	*Cover.*	*MDD*	*Return*	*Romad*
ADADELTA	0.5	TanH	**0.58**	0.38	0.48	0.20	**38.75**	362.75	**9.36**
ADADELTA	0.3	Relu	0.53	0.59	0.55	0.77	79.50	520.25	6.54
ADAM	0.9	Selu	**0.58**	0.57	0.57	1.00	135.75	**833.50**	6.14
RMSPROP	0.3	Selu	0.53	**0.74**	**0.60**	0.42	121.00	541.00	4.47
ADAGRAD	0.5	Selu	0.20	0.62	0.50	0.14	54.25	223.75	4.12
ADAM	0.5	Relu	0.55	0.52	0.53	0.80	151.25	598.25	3.96
ADAGRAD	0.7	Sigmoid	0.57	0.53	0.55	0.86	200.50	610.50	3.04
RMSPROP	0.7	Sigmoid	0.52	0.50	0.51	1.00	134.75	386.00	2.86
ADAGRAD	0.3	Linear	0.50	0.56	0.53	0.34	117.25	321.50	2.74
ADAM	0.3	Sigmoid	0.57	0.55	0.56	0.41	115.75	300.50	2.60

At first, we observe that the Adadelta optimizer presented the two best positioned results in terms of Romad, by using, respectively, 50% of explorations with TanH activation and 30% of explorations with Relu activation. The TanH activator presented a return that is 9.36 higher than the risk assumed in trading, but it trades only 20% of the time. Such a strategy showed the lowest risk, illustrated by a maximum drawdown of 38.75. It also showed the best long precision (58%), which entails its very well position in terms of the Romad metric. The Adam optimizer showed three configurations positioned in the top-10 approaches. In particular we noticed that, when using a very high probability of explorations (90%) and the Selu activation, it increased the profit (the highest of all, with 833 market points), but also the risk (an MDD of 135.75, the third highest of the experiments). Even in that scenario, it showed a Romad of 6.14, the third best of all. However, that comes with a 100% coverage, which means that the agent was more exposed at the market risk.

Conversely, the Rmsprop activation function had only one configuration in the Romad top-10 results, however, it showed the best accuracy of all (60%), and also the best precision of shorts (a surprising 74%). However, that high short precisions do not yield a very high return in the market period analyzed, specially considering that the agent trades only 42% of the time. However, that configurations still had a good risk-return trade-off, with a 4.47 Romad. Finally, Adagrad optimization approach had three configurations in the top-10 results. Its best configuration uses 50% probability of explorations in the RL agent training, with the Selu activation approach. However, it showed the lowest long precision and accuracy of all approaches, but its fifth best position in that table is mainly due to the fact that such an agent enters the market only 14% of the time (second lowest of all), but also providing the second lowest risk of all the approaches.

6.3.2 Experimental Results and Comparison With Baselines

We now focus on the comparison of our proposed technique against the baselines previously summarized in Section 6.2.2. As previously outlined, we start our analysis of the experimental results by comparing the performance of our multi-ensemble and multi-layer approach, against several well-performing variations of it. These variations only rely on the first layer of the stack, thus not providing the reinforcement learning components and ensembles. However, the experiments show how our second and third levels based on RL both help to significantly improve the overall performances of the approach.

To obtain such a result, we considered, for all the strategies, a real-world trading scenario where we assumed to trade a single future on the S&P 500 market, with the same initial investment capital of 4,000 market points (200,000 USD). As a robust training period, we consider the same 7-years period of Section 5.1, starting from February 1st of 2012 to September 5 of 2018, while as test set, we exploited a time frame spanning September 6th of 2018 to August 30th of 2019 (the last trading year of our dataset). Note that it represents an interesting period to analyse, since it is characterized by a first phase in which the market suffered high volatility and a significant fall, mainly due to commercial war between the USA and China, opposed to a second phase, in which the market experienced a strong recovery, with very positive performance. Table 6.3 shows the results of our experiments.

Table 6.3: Summary of the comparison between our RL layers and the non-RL ensembling techniques outlined in Sec. 6.2.2, in an out-of-sample trading scenario (September 2018 to August 2019). Best results per trading metric are underlined. Trading metrics are expressed in market points (1 point = 50 USD) and classification metrics are in the interval [0,1].

	Classification metrics			*Trading metrics*		
Method	**Long prec.**	**Short prec.**	**Cover.**	**MDD**	**Return**	**RoMaD**
RL-ensemble *(proposed)*	0.59	0.50	1.00	221.74	**<u>1265.50</u>**	**<u>5.70</u>**
Majority ensemble	0.60	0.69	0.55	162.75	901.00	5.53
Thresh. voting ensemble (50%)	0.61	0.60	0.35	**<u>130.75</u>**	662.75	5.06
Thresh. voting ensemble (70%)	0.31	0.00	0.06	186.74	-89.24	-0.47
Buy-and-Hold baseline	-	-	-	598.00	45.50	0.08

First, let us analyse the Buy-and-Hold strategy, which represents a passive investment approach, commonly used as a trading benchmark since it essentially replicates the market behaviour. In our test period, B&H performances return a modest income, but with a high investment risk (due to the market fall at the end of 2018). Since S&P500 is a usually growing market, the Buy-and-Hold strategy is generally effective, but it is in such periods that it shows its limits. This is particularly relevant when operating with the *leverage*, in which sudden and sharp falls can lead to a loss of the invested capital, and therefore require to provide new investments to keep the position with the broker active (*margin calls*). Overall, it is therefore very important that advanced trading strategies should be able not to follow the market trend during negative phases, while, conversely, to detect and exploit the positive trends.

For this reason, when looking at the results in Table 6.3, we did not take into close consideration the results in terms of classification (*e.g., precisions*), and in particular we did not evaluate them in terms of accuracy. This is because, compared to a standard classification task, in trading it is not important the amount of correct choices, but rather to correctly select the operations with a higher specific weight (*i.e.,* days with higher volatility), since a single wrong choice can compromise the result of a high number of correct choices. On the other hand, we give particular emphasis to the most significant indicators in the analysis of the financial strategy, i.e. the return, the MDD and – specifically – the RoMaD, which quantifies the ratio between the obtained profit and the assumed risk, thus making it possible to determine the safest and most robust solution to adopt.

Looking at Table 6.3, we can summarize the results of the competitors of the proposed approach as it follows. The strongest baseline is represented by an ensemble of independent neural networks, obtained as the majority voting of their decisions. This "naive" ensemble is therefore performed over the first layer of our approach. On the practical side, it shows to be a very effective strategy, with an extremely positive behaviour which overperforms the market throughout the considered test period. It provides the second highest return (901 market points, or 45, 050.00 USD), the second lowest MDD and also the second highest RoMaD of 5.53. However, this agent is very cautious about its decisions, not entering the market in almost half of the trading days (i.e. coverage is 0.55)

The rest of the approaches presented in the third and fourth row of Table 6.3 represent two variations of the aforementioned first-layer majority voting ensemble. These variations are obtained by imposing a minimum agreement threshold on the majority decision. However, from our experiments, this technique has shown to cause a regular decrease of the RoMaD as the threshold used increases. Therefore, we only reported results with agreement thresholds of 50 and 70%, for illustrative purposes. Going into detail, with a 50% decision agreement, the trader agent enters the market 35% of the days, providing the third highest return, which, according to its RoMaD, is 5.06 times higher than the risk. Notably, this solution achieves the lowest maximum drawdown value, thanks to a balanced combination of low coverage and high precision of operations, making it an effective and low-risk strategy for the considered period. Conversely, the ensemble with a 70% of majority voting agreement gives the overall worst performances, with a negative return and – consequently – a negative RoMaD. This approach is useful to only show how a reduction in coverage does not automatically translate into a decrease in risk, and also to denote the existence of a non-negligible variance in the predictions of individual CNNs.

At last, we analyse the results of our proposed approach, based on a metalearner that autonomously learns how to ensemble the CNN decisions. The metalearner is then executed in separate iterations, and its decisions are subjected to a second level of majority ensembling (in order to balance the different first-level ensembling strategies found by the metalearner). Such a multi-ensemble method provides the best return so far (1265 market points, equivalent to 63,250.00 USD, which means 5,270.83 USD per month in a very difficult period). Even though such a profit came with the second highest MDD of all (221.74 market points), it is still more than a half the value of the B&H strategy, and only the 0.055% of the initial investment (4,000 market points), which can still be considered a modest risk for a financial strategy.

Notwithstanding, even in this scenario, we increased our best competitor return by 28.78%, also providing the highest RoMaD of 5.70. In other words, this means that the given return is almost six times the value of the risk assumed. To better understand how our approach boosts the best baseline (first-layer majority voting ensemble), in Figure 6.6 we show their equity curves, together with the Buy-and-Hold baseline that, as previously said, indicates the market behaviour.

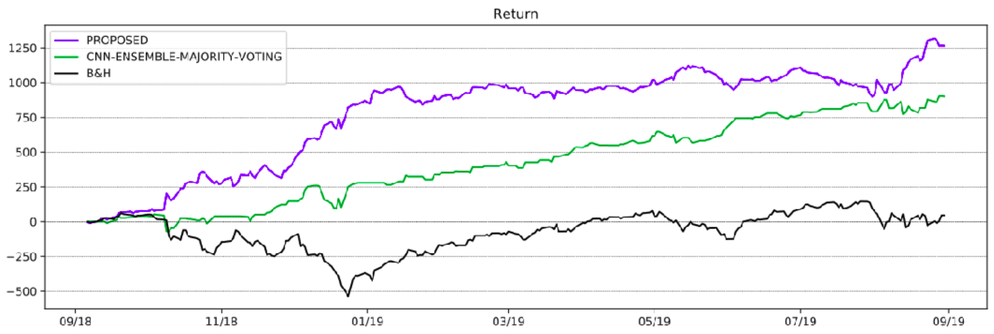


Figure 6.6: Equity curve of our RL-ensemble approach, against the best non-RL baseline (majority voting ensemble) and the Buy-and-Hold strategy.

According to Figure 6.6, the proposed approach is not only consistently above, in terms of return, both to the market and to the competitor throughout the period considered, but also appears to be more effective in generating profit during negative market phases. Moreover, it is the only one of the three strategies to have a consistently positive return over the period in question. Following these results, we can therefore say that the adoption of the second layer based on the RL-ensemble technique is promising in terms of effectiveness of the final trading strategy provided, and seems to globally confirm the validity of the proposed approach.

Finally, to conclude the analysis of the performance of our approach, we start the comparison of our method against the last two baselines discussed in Section 6.2.2: the approach proposed by Sezer *et al.* in [150] (*CNN-TAr*), and the one by Calvi *et al.* [151] (*LS-STM*), which represents two different strong competitors in literature. To do that, we consider new symbols (MSFT and JPM) and periods, with respect to previous experiments, that has been used in the considered works.

Table 6.4: Comparison of the real-world trading performance, in terms of annualized returns, between our method and some literature competitors when considering different periods and markets. Our method clearly outperforms all the competitors in the considered settings.

[h!]

	Annualized return		
	Period: 2007-2012		*Period: 2009-2015*
	JPM stock	MSFT stock	$S\&P500$ future
OUR PROPOSAL	**<u>11.3%</u>**	**<u>17.53%</u>**	**<u>23.20%</u>**
CNN-TAr [150]	9.19%	4.44%	*n.a.*
LS-STM [151]	*n.a.*	*n.a.*	$\sim$ 10.15% 1
Buy-and-Hold	-1.67%	-1.93%	21.33%

The annualized return value is approximate, since the authors in [151] reported the cumulative profit chart only.

Table 6.4 reports the results achieved in terms of annualized return, i.e. the percentage of annual return on the initial investment (which we assumed to be equal to the stock/future value at the begin of the period), together with the Buy-and-Hold performance, which allows us to estimate the behaviour of the market. The first period we considered spans the date interval between January 1st of 2007 and December 31st: in these six years, both JPM and MSFT recorded a slightly negative trend, mainly due to the sever economic crisis of 2008, associated with sub-prime mortgages. However, both our proposal and the CNN-TAr approach manage to generate a positive return in this phase: in particular, our approach has proved to be comparable and even mildly superior when trading on the JPM symbol, while it showed to be extremely performing against this competitor for what concerns the MSFT index.

On the other hand, with regard to the second period, which covers dates between January 1st of 2009 and December 31st of 2015, we examine the performance of our approach when trading on the S&P500 future against the LS-STM competitor. We notice that the achieved annualized return of the latter is approximated, since in the original work from Calvi et al. [151], the authors plotted the cumulative profit curve without specifying the punctual values of returns. However, our method clearly outperforms this competitor, and also the Buy-and-Hold strategy, which nevertheless highlights a very strong market trend in this period, hard to outrun.

Overall, both in relation to the non-RL ensemble techniques we used as benchmarks and to the competitors present in literature, in the periods and markets they covered, our approach showed to be competitive in performance and robust in results. As outlined in next Section **??**, further studies are needed to investigate the response of our method in markets with different behaviours and characteristics, as well as for the management of compound portfolios.

Chapter 7

Hawkeye: a Visual Framework for Agile Cross-Validation of Deep Learning Approaches in Financial Forecasting

7.1 Introduction

As mentioned before, Financial Forecasting represents a challenging task, mainly due to the irregularity of the market, high fluctuations and noise of the involved data, as well as collateral phenomena including investor mood and mass psychology. Among these, it is worth to mention the bias introduced by the choice of model weights initialization and the considered observation periods, as well as the narrow separation between significant results and noise, typical of the financial domain. A thorough analysis of these peculiar issues lead to a substantial increase of the experiments and results to analyze, making the discovery of meaningful hidden patterns very difficult and time consuming to perform. To cope with these concerns, in this chapter we propose a visual framework for in-depth analysis of results obtained from Deep Learning approaches, in order to predict the daily price variation of Financial Markets, along with evaluating the robustness and predictive power of the models used. Our framework offers a modular view, both general and targeted, of results data, several instruments to easily navigate over different walks, classifiers and training *epochs*, a large set of metrics (e.g., Sharpe Ratio, Precision, Maximum Drawdown, Accuracy, etc.) and plots to assess both the economic performance and the goodness of the classification process, and an advanced system for custom reports generation and sub-period investigation.

The general architecture of our proposed framework is depicted in Figure 7.1. First, we show how the input data are organized and handled;

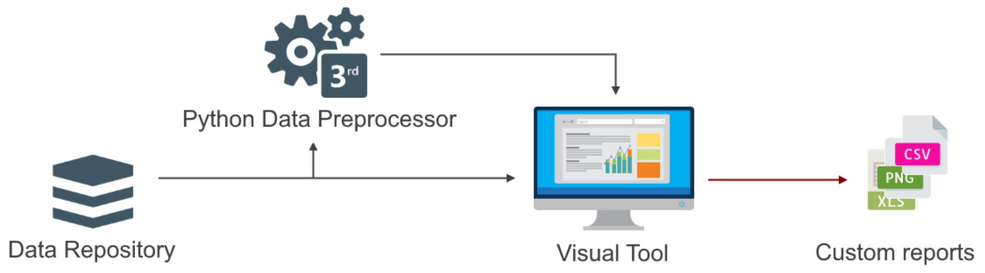


Figure 7.1: The high-level architecture of our framework.

then, we describe more specifically the core components of the tool. Notably, a demo version of the framework, with preset sample data, is available at `http://hawkeye.unica.it`, and publicly accessible by using `Hawkeye` and `madain_demo` as login credentials.

7.2 The Proposed Framework

7.2.1 Data Organization & Pre-processing

The first two fundamental components of our framework are the Data Repository (DR) and the Data Preprocessor (DP). The DR is a storage component, i.e. a repository which contains a directory $\texttt{res}_i$ for each different results scenario i to analyze. Within $\texttt{res}_i$, the user must upload results data files in a custom data structure, compliant with our framework. In particular, by default, $\texttt{res}_i$ would contain: (i) a `log.json` file, which contains general information about the analysed scenario and the used Deep Learning parameters (e.g., number of epochs, time period considered, selected loss function, learning rate, *etc.*. An example can be seen here[1]); and, (ii) two main folders, one for `validation` data and one for `test` data (if provided), internally partitioned into *walks*, in order to reproduce the walk-forward subdivision. Each `walk_j` sub-folder then provides a list of `results_k.json` files, for every Deep Learning classifier tested on the considered walk. An example of the two folders can be seen here[2].

This repository structure, although general enough to integrate with any classification tool based on Deep Learning, is not restrictive, since the framework also provides a specific module (which acts as a middle layer) to easily adapt other repository structures and formats to import data in our tool.

On the other hand, the DP is a software component, implemented in Python 3, responsible for an *on-the-fly* processing of input data, required to

[1] `http://hawkeye.unica.it/log_demo`
[2] `http://hawkeye.unica.it/zip_demo`

generate specific views or custom comparisons, or to simulate and modify, at runtime, several real-world trading parameters such as stop-loss or broker fees.

7.2.2 Visual Tool

We now describe how the core component of our framework, the Visual Tool (VT), works. It consists of a Graphic User Interface (GUI) to deeply analyze the results of different Deep Learning classification scenarios where the walk-forward cross-validation strategy is employed to solve financial classification forecasting problems. As previously mentioned, since randomness severely affects the results in this specific domain, the VT is designed to ease the navigation and scrolling through the results, for different walks (i.e., market periods) and epochs (i.e., model iterations).

Once logged in, a screen with a list of imported scenarios is shown. Then, by selecting a specific scenario, the tool loads its main dashboard, which looks as illustrated in Figure 7.2. It exhibits three functional blocks, described in the following.

Toolbar. The first block (in red) implements a toolbar which let the user navigate, explore and filter the loaded scenario. For instance, it is possible to switch between the validation and test views or to show the average behavior of all classifiers, in order to investigate the model robustness. Moreover, it makes easier to zoom over specific ranges of epochs, as shown in Figure 7.3, to accurately and punctually check the performance of a single classifier, and to exclude selectively models subject to over-fitting. All the choices made within this toolbar affect the plots visualization in the third block, described next.

Summary box. The second block (in green) simplifies the navigation between different scenarios and views. Indeed, the VT provides keyboard shortcuts to quickly move between scenarios and parameters. Through the summary box, the analyst can therefore constantly monitor the displayed view.

Plot area. The third and last functional block (in cyan) represents the key part of our VT. It provides a dynamic plot for each different metric featured by the framework. Most of the displayed metrics relate to the assessment of financial performance, and can be informally summarized as follows: (i) *Return*[3], i.e. the actual profit achieved at the end of the period taken into account; (ii) *Sharpe Ratio* [153], a common statistical measure of financial

[3]https://www.investopedia.com/terms/r/return.asp

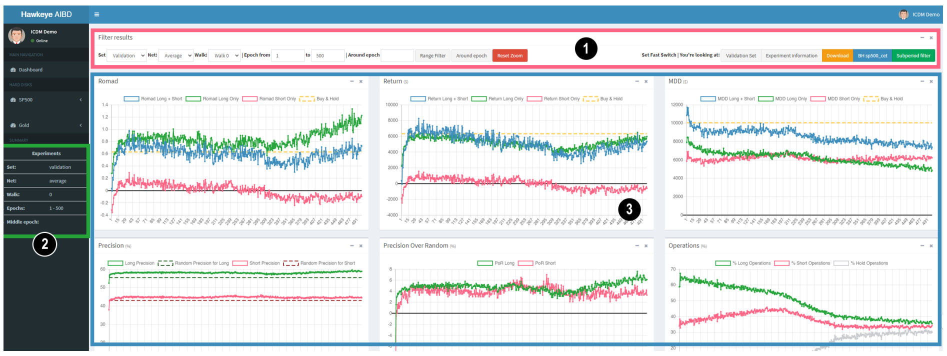


Figure 7.2: Main dashboard of the VT. It features: (1) the toolbar; (2) the summary box; and, (3) the plot area.

performance, given by the average return earned in excess of a risk-free strategy (e.g., the passive income generated by a *U.S. Treasury Bill* rate in the same period), divided by the average volatility of the portfolio (i.e., the standard deviation of returns); (iii) *Sortino Ratio* [154], similar to the previous Sharpe Ratio, except that it only takes into account the downside volatility (i.e., the standard deviation of negative returns only); (iv) *Maximum Drawdown* [155] (MDD), which is intuitively defined as the highest difference (in dollars) between a local maximum point and a subsequent local minimum point in the profit curve, or the maximum accumulated slump (commonly used as an alternative estimator of the investment risk); and, (v) *Return over Maximum Drawdown*[4] (Romad) that is, as the name itself suggests, the ratio between Return and MDD, which collectively estimates the profitability of the trading strategy.

Along with these financial indicators, the plot area also provides standard metrics for the analysis of the Deep Learning process, in particular: (i) *Precision*, i.e. the percentage of correctly classified items of a given class, over all the instances of that class; (ii) *Precision-over-Random* (PoR), to estimate how far the computed precision deviates from a benchmark given by the random predictor; (iii) *Coverage*, which defines the percentage of operations performed by the classifier (between `long`, `short` and `idle` actions); (iv) *Accuracy*, which is the ratio between the number of correct predictions over the total number of samples; and, (v) *Loss*, i.e the values of the *loss function* used to during the training of the classifier, which usually represents the classification error for the considered sample set (e.g., root-mean-square error, binary cross-entropy, *etc.*).

Furthermore, when solving Financial Forecasting problems trough Deep Learning techniques, a common practice is that of specializing some networks/classifiers in specific trading policies, depending on the general be-

[4] `https://tinyurl.com/hawkeye-romad`

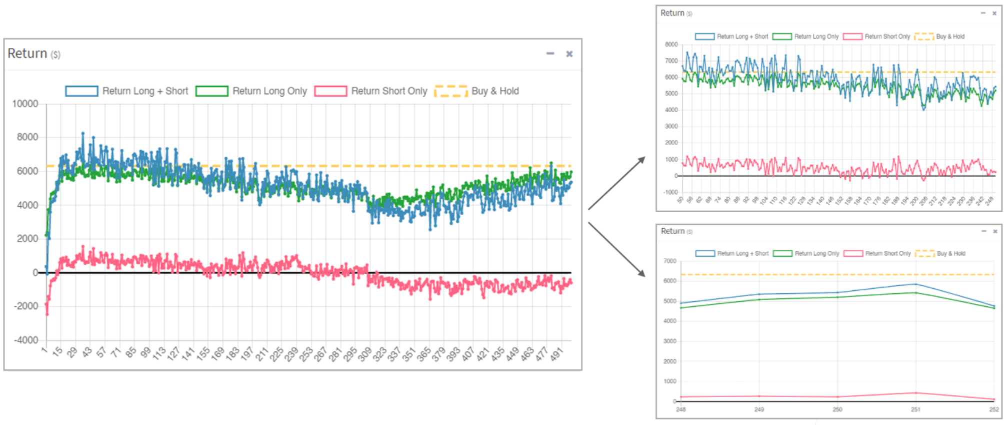
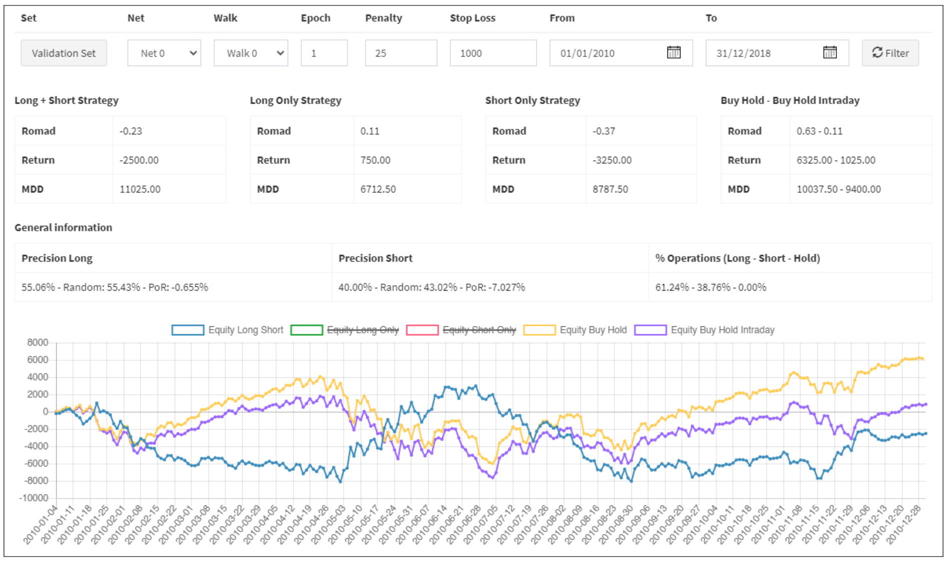


Figure 7.3: Different display modes of the *plot area*: full epoch range (on the left), custom range selection (on the upper right) and single epoch zoom (on the lower right).



Set	Net	Walk	Epoch	Penalty	Stop Loss	From	To	
Validation Set	Net 0	Walk 0	1	25	1000	01/01/2010	31/12/2018	Filter

Long + Short Strategy

Romad	-0.23
Return	-2500.00
MDD	11025.00

Long Only Strategy

Romad	0.11
Return	750.00
MDD	6712.50

Short Only Strategy

Romad	-0.37
Return	-3250.00
MDD	8787.50

Buy Hold - Buy Hold Intraday

Romad	0.63 - 0.11
Return	6325.00 - 1025.00
MDD	10037.50 - 9400.00

General information

Precision Long	Precision Short	% Operations (Long - Short - Hold)
55.06% - Random: 55.43% - PoR: -0.655%	40.00% - Random: 43.02% - PoR: -7.027%	61.24% - 38.76% - 0.00%

Figure 7.4: The sub-period filter.

haviour of the target market. From this perspective, the VT also offers the possibility of selectively displaying the metrics for three different trading policies: `long+short`, `long` only and `short` only. Finally, each plot also provides a comparison with the Buy&Hold benchmark, in the same considered period.

Sub-period filter. In addition to the three functional blocks of the main dashboard, the VT provides some additional features, accessible through the toolbar. Indeed, analyzing the global results of the classifiers may be not sufficient, as it does not give clear indications of day-by-day performance. To tackle this problem, the VT exploits the preprocessor to generate, at run-time, the *equity curve* (i.e., the cumulative profit curve) of a deep classifier, for a specific walk and epoch. This curve, along with summary information and benchmark data, is then shown in the *sub-period filter* popup, as illustrated in Figure 7.4. Through this popup, the user may also apply a custom broker fee (in the *penalty* field), specify a stop-loss threshold, as well as select a sub-period of observation.

Custom reports. Finally, starting from the input data, the framework generates advanced reports and offers the user the possibility to download them locally, for offline analysis and sharing. Through this feature, the user can download on his/her computer several types of documents and files, including: (i) the rasterized `png` version of the plots displayed in the VT; (ii) the `hdf5` snapshot of the Deep Learning models used (we have tested our framework with Keras); (iii) the `csv` files containing day-by-day predictions; and (iv) the `xls` reports, which notably provide detailed data and results for each scenario, as well as targeted threshold analyses.

Chapter 8

Conclusions

In this thesis, the use of Machine Learning and Deep Learning has been applied in the Finance domains through not yet experimented methodologies and novel approaches.

Machine Learning techniques cover a crucial role in many financial contexts since they are used to evaluate the potential risks of investments, thus reducing the losses due to unreliable strategies. The high degree of complexity, dynamism and non-stationary nature of data involved in the futures markets makes the definition of an effective prediction model a challenge.

In order to face this problem, this dissertation introduced an auto-configurable ensemble approach to significantly improve the trading performance. This is done through optimizing two sets of parameters in late and early past data, returning customized ensembles that act by complete agreement strategy in any kind of market. By following a methodological experimental approach, our proposed automatic ensemble method starts auto-tuning hyper-parameters in late past data. Among these hyper-parameters, we tune *feature selection* parameters, which will return the best possible inputs for each classifier in the ensemble and also *time series* parameters, which will present the best disposal of features for the classifiers. As the last step, we tuned intrinsic-parameters, creating powerful ensembles with classifiers trained with recent past data. Such an automatic ensemble fine-tune model returns an ensemble of the best possible individual classifiers found in the training data, which can be applied for different markets. All these parameters optimizations are done through a *Walk Forward Optimization* approach considering the *non-anchored* modality. Results of trading in an out-of-sample data, spanning years *2016*, *2017*, and *2018* show that, despite the data complexity, the proposed ensemble model is able to get good performance in the context of positive returns in all the markets considered, being a good strategy for conservative investors who want to diversify, but keeping their investments profitable. It also turned out that in one market the proposed approach fails in achieving better trading performance than benchmarks. We believe that, in addition to select ad-hoc hyper-parameters and intrinsic-parameters for each market, an automatic selection of new ad-hoc classifiers to be used by our proposed approach must also be done. We believe that this additional step can find new classifiers useful to understand different natures of data from different markets.

Secondly, we have proposed an innovative approach for the forecasting of market behavior by using Deep Learning technologies and by encoding time series to GAF images. The developed CNNs have been applied to the GAF images for a classification task. Moreover, an ensemble was fed with the CNNs above a majority voting strategy that has been used to select the final classification. High results have been obtained using the S&P500 Future, the market where we have trained, validated and tested our networks and the overall ensemble. The GAF imaging technique has thus been applied within the financial technology domain bringing the benefits of CNN. Moreover, our approach, a combination of Deep Learning and GAF images technologies, outperformed the benchmarks strategies consisting of Buy-and-Hold operations for a classification task where long and short actions can be performed. The analysis of the tuning of several hyper-parameters is being carried out by our team and is subject of future works. Currently, we are studying the stacking policy of the GAF images (as shown in Fig.5.2), and how accuracy and net profit vary according to the value of K parameter (currently set to 4). Our approach outperforms the Buy-and-Hold strategy, although the latter was still very competitive and profitable within the considered period.

As a third novel approach, in this dissertation we have proposed a step forward in efficient stock trading with ensembles by presenting an approach that uses two well-known and efficient Machine Learning approaches, namely Deep Learning and Deep Reinforcement Learning, in a three-layer fashion. The proposed method then exploits several ensembling steps to provide its final intra-day trading strategy: firstly, we stack hundreds of Deep Learning decisions, provided by a large number of CNNs trained with historical market data encoded into GAF images, and use them as input for a Reinforcement Meta Learner Classifier; and, lastly, we ensemble the decisions of several training iterations of the meta learner.

From our experimental results, we observe that: (i) the meta learner leads to better trading results and less overfitting when we preliminary explore the training parameters and adopt the most promising ones; (ii) compared to well-performing non-RL ensemble benchmarks, our approach showed a final return improvement of 28.78% when compared to our best benchmark (which, notably, has still very good performances in the considered period), by yielding returns 5.70 higher than the risk assumed, and outperforming the market not only in the crisis scenario of 2018, but also in the following phase of 2019 in which the market showed a very solid trend; and, finally, (iii) our approach provides the best performances, when tested against the strong methods proposed by Sezer et al. [150] and Calvi et al. [151], in two different real-world trading scenarios, and when considering several distinct periods and markets.

However, the analysis of the results coming from Deep Learning approaches applied to Financial Forecasting problems has proven to be a very difficult task, mainly due to the complexity in determining the predictive power and robustness of the models used. In fact, the large number of factors that influence the market behavior, as well as the importance of (i) identifying significant and meaningful patterns, (ii) selecting representative analysis periods and, above all, (iii) filtering inaccurate or over-fitting outcomes, may represent complex issues even for experienced researchers and data analysts. To address these issues, in this thesis we have proposed a visual framework for in-depth analysis of results obtained from Deep Learning classifiers specialized in the financial domain. Notably, our framework is not limited to meet only the aforementioned requirements, but it also offers advanced metrics for measuring both economic performance and quality of the classification models adopted, along with targeted tools for exploring the results and generating comprehensive reports. Furthermore, the platform has shown to be effective already for real-world usage, as it is currently adopted and exploited by a *fintech* company for the optimal tuning and in-depth analysis of several robo-trading systems, representing its main core business.

8.1 Future Research Directions

Our research on Machine Learning and Deep Learning applied to the Financial Forecasting domain has led to different novel approaches but still poses some interesting challenges that require further investigation. One more path we are already exploring consists of applying the results of our ensembles approaches to real trading platforms. The goal is, on the one hand, to simulate the real earnings we would have obtained on the past data and the desired market. With the test being robust, on the other hand, the next step would be to perform real-time tradings on a certain number of markets. The platform we are already playing with is MultiCharts[1]. Moreover, one more possible future work can involve the definition of multi-market strategies, able to improve the prediction performance by diversifying the investments or by using information about the behavior of many markets, in order to fine-tune the kind of classifiers used or their predictions. Finally, as stated before, a *data-driven* selection of classifiers for the ensemble, rather than just intrinsic and hyperparameters, is a promising research direction to be done.

Regarding the algorithm proposed in Chapter 5, we would also like to test our approach on different classification tasks within several domains such as

[1] `https://www.multicharts.com/`

Sentiment Analysis, Emotion Detection, Credit Scoring. The reason is to understand how GAF imaging performs in presence of text or different kinds of feature vectors.

Furthermore, we aim to extend the network structure proposed in Chapter 6 in order to reduce the effects of overfitting: for instance, we think that adding recurrent layers – such as Long Short Term Memories networks – in the proposed pipeline can potentially help to improve the results. Additionally, we also aim to evaluate the impact of optimization algorithms [156, 157] when training both the first and second layers of our approach. And, finally, we would focus our future investigations on the fusion of different meta learner optimizers/parameters, which may increase the number of experts in the final ensembling methodology and, therefore, lead to a more robust and stable trading strategy.

For what concern Hawkeye, the visual Framework proposed in Chapter 7 we are evolving the system towards several directions, focussing in particular on: (i) adding more plots and metrics to provide a more detailed analysis and investigation of results; (ii) extend our Framework to support models based on regression in addition to those based on classification; and, finally, (iii) generalizing the system to domains beyond the financial and economic contexts.

Bibliography

[1] S. Carta, A. Corriga, A. Ferreira, D. Recupero, and R. Saia, “A holistic auto-configurable ensemble machine learning strategy for financial trading,” *Computation*, vol. 7, no. 4, pp. 1–25, 2019.

[2] S. Barra, S. Carta, A. Corriga, A. Podda, and D. Recupero, “Deep learning and time series-to-image encoding for financial forecasting,” *IEEE/CAA Journal of Automatica Sinica*, vol. 7, no. 3, pp. 683–692, 2020.

[3] S. Carta, S. Consoli, A. Corriga, R. Dapiaggi, A. Podda, and D. Reforgiato Recupero, “Hawkeye: A visual framework for agile cross-validation of deep learning approaches in financial forecasting,” 2020.

[4] S. Carta, A. Corriga, A. Ferreira, A. Podda, and D. Recupero, “A multi-layer and multi-ensemble stock trader using deep learning and deep reinforcement learning,” *Applied Intelligence*, vol. 51, no. 2, pp. 889–905, 2021.

[5] G. G. Calviand V. Lucicand D. P. Mandic, “Support tensor machine for financial forecasting,” in *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 8152–8156, May 2019.

[6] R. C. Cavalcante, R. C. Brasileiro, V. L. F. Souza, J. P. Nóbrega, and A. L. I. Oliveira, “Computational intelligence and financial markets: Asurvey and future directions,” *Expert Systems with Applications*, vol. 55, pp. 194–211, 2016.

[7] T. Kimoto, K. Asakawa, M. Yoda, and M. Takeoka, “Stock market prediction system with modular neural networks,” in *Neural Networks, 1990., 1990 IJCNN International Joint Conference on*, pp. 1–6, IEEE, 1990.

[8] T. Oberlechner, “Importance of technical and fundamental analysis in the european foreign exchange market,” *International Journal of Finance & Economics*, vol. 6, no. 1, pp. 81–93, 2001.

[9] H. V. Roberts, "Stock-market "patterns" and financial analysis: Methodological suggestions," *The Journal of Finance*, vol. 14, no. 1, pp. 1–10, 1959.

[10] A. S. Weigend, *Time series prediction: forecasting the future and understanding the past.* Routledge, 2018.

[11] C. Abad, S. A. Thore, and J. Laffarga, "Fundamental analysis of stocks by two-stage dea," *Managerial and Decision Economics*, vol. 25, no. 5, pp. 231–241, 2004.

[12] G. A. Griffioen, "Technical analysis in financial markets," 2003.

[13] G. Preethi and B. Santhi, "Stock market forecasting techniques: A survey.," *Journal of Theoretical & Applied Information Technology*, vol. 46, no. 1, 2012.

[14] J. Patel, S. Shah, P. Thakkar, and K. Kotecha, "Predicting stock and stock price index movement using trend deterministic data preparation and machine learning techniques," *Expert Systems with Applications*, vol. 42, no. 1, pp. 259–268, 2015.

[15] X. Ding, Y. Zhang, T. Liu, and J. Duan, "Deep learning for event-driven stock prediction," in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[16] T. H. Nguyen and K. Shirai, "Topic modeling based sentiment analysis on social media for stock market prediction," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, vol. 1, pp. 1354–1364, 2015.

[17] J. Bollen, H. Mao, and X. Zeng, "Twitter mood predicts the stock market," *Journal of computational science*, vol. 2, no. 1, pp. 1–8, 2011.

[18] T. Rao and S. Srivastava, "Analyzing stock market movements using twitter sentiment analysis," in *Proceedings of the 2012 international conference on advances in social networks analysis and mining (ASONAM 2012)*, pp. 119–123, IEEE Computer Society, 2012.

[19] S. Carta, A. Corriga, R. Mulas, D. R. Recupero, and R. Saia, "A supervised multi-class multi-label word embeddings approach for toxic comment classification," in *KDIR*, pp. 105–112, ScitePress, 2019.

[20] D. Dessì, F. Osborne, D. R. Recupero, D. Buscaldi, and E. Motta, "Generating knowledge graphs by employing natural language processing and machine learning techniques within the scholarly domain," *Future Generation Computer Systems*, vol. 116, pp. 253–264, 2021.

[21] F. Hoppe, D. Dessì, and H. Sack, "Deep learning meets knowledge graphs for scholarly data classification," in *Companion Proceedings of the Web Conference 2021*, pp. 417–421, 2021.

[22] S. Soni, "Applications of anns in stock market prediction: a survey," *International Journal of Computer Science & Engineering Technology*, vol. 2, no. 3, pp. 71–83, 2011.

[23] T. Lintonenand T. Räty, "Self-learning of multivariate time series using perceptually important points," *IEEE/CAA Journal of Automatica Sinica*, vol. 6, pp. 1318–1331, November 2019.

[24] K. Żbikowski, "Using volume weighted support vector machines with walk forward testing and feature selection for the purpose of creating stock trading strategy," *Expert Systems with Applications*, vol. 42, no. 4, pp. 1797–1805, 2015.

[25] S. Chatterjee and A. S. Hadi, *Regression analysis by example.* John Wiley & Sons, 2015.

[26] Y. Zhang and L. Wu, "Stock market prediction of s&p 500 via combination of improved bco approach and bp neural network," *Expert systems with applications*, vol. 36, no. 5, pp. 8849–8854, 2009.

[27] D. Zhang, M. Hu, and Q. Ji, "Financial markets under the global pandemic of covid-19," *Finance Research Letters*, p. 101528, 2020.

[28] J. W. Goodell, "Covid-19 and finance: Agendas for future research," *Finance Research Letters*, p. 101512, 2020.

[29] T. Z. Tan, C. Quek, and G. S. Ng, "Brain-inspired genetic complementary learning for stock market prediction," in *2005 IEEE Congress on Evolutionary Computation*, vol. 3, pp. 2653–2660, IEEE, 2005.

[30] E. A. Gerlein, M. McGinnity, A. Belatreche, and S. Coleman, "Evaluating machine learning classification for financial trading: An empirical approach," *Expert Systems with Applications*, vol. 54, pp. 193–207, 2016.

[31] S. Scardapane and D. Wang, "Randomness in neural networks: an overview," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 7, no. 2, p. e1200, 2017.

[32] P. Misra *et al.*, "Machine learning and time series: Real world applications," in *Computing, Communication and Automation (ICCCA), 2017 International Conference on*, pp. 389–394, IEEE, 2017.

[33] H. Ince and T. B. Trafalis, "A hybrid forecasting model for stock market prediction.," *Economic Computation & Economic Cybernetics Studies & Research*, vol. 51, no. 3, 2017.

[34] L. A. Teixeira and A. L. I. De Oliveira, "A method for automatic stock trading combining technical analysis and nearest neighbor classification," *Expert systems with applications*, vol. 37, no. 10, pp. 6885–6890, 2010.

[35] V. P. Upadhyay, S. Panwar, R. Merugu, and R. Panchariya, "Forecasting stock market movements using various kernel functions in support vector machine," in *Proceedings of the International Conference on Advances in Information Communication Technology & Computing*, p. 107, ACM, 2016.

[36] R. Hafezi, J. Shahrabi, and E. Hadavandi, "A bat-neural network multi-agent system (bnnmas) for stock price prediction: Case study of dax stock price," *Applied Soft Computing*, vol. 29, pp. 196–210, 2015.

[37] U. N. Chowdhury, S. K. Chakravarty, and M. T. Hossain, "Short-term financial time series forecasting integrating principal component analysis and independent component analysis with support vector regression," *Journal of Computer and Communications*, vol. 6, no. 03, p. 51, 2018.

[38] B. Vanstone and G. Finnie, "An empirical methodology for developing stockmarket trading systems using artificial neural networks," *Expert systems with Applications*, vol. 36, no. 3, pp. 6668–6680, 2009.

[39] T. Rollinger and S. Hoffman, "Sortino ratio: A better measure of risk," *Futures Magazine*, vol. 1, no. 02, 2013.

[40] J. White and V. Haghani, "A brief history of sharpe ratio, and beyond," *Available at SSRN 3077552*, 2017.

[41] A. Frugier, "Returns, volatility and investor sentiment: Evidence from european stock markets," *Research in International Business and Finance*, vol. 38, pp. 45–55, 2016.

[42] M. Haenlein and A. Kaplan, "A brief history of artificial intelligence: On the past, present, and future of artificial intelligence," *California management review*, vol. 61, no. 4, pp. 5–14, 2019.

[43] M. Marras, “Machine learning models for educational platforms,” 2020.

[44] I. Muhammad and Z. Yan, “Supervised machine learning approaches: A survey.,” *ICTACT Journal on Soft Computing*, vol. 5, no. 3, 2015.

[45] R. Saravanan and P. Sujatha, “A state of art techniques on machine learning algorithms: a perspective of supervised learning approaches in data classification,” in *2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS)*, pp. 945–949, IEEE, 2018.

[46] D. Dessi, “Knowledge extraction from textual resources through semantic web tools and advanced machine learning algorithms for applications in various domains,” 2020.

[47] L. Breiman, “Random forests,” *Machine Learning*, vol. 45, pp. 5–32, Oct 2001.

[48] S. Carta, A. S. Podda, D. R. Recupero, R. Saia, and G. Usai, “Popularity prediction of instagram posts,” *Information*, vol. 11, no. 9, 2020.

[49] L. Deng, “A tutorial survey of architectures, algorithms, and applications for deep learning,” *APSIPA Transactions on Signal and Information Processing*, vol. 3, 2014.

[50] M. Magdon-Ismail and A. F. Atiya, “Maximum drawdown,” *Risk Magazine*, vol. 17, no. 10, pp. 99–102, 2004.

[51] H. Alostad and H. Davulcu, “Directional prediction of stock prices using breaking news on twitter,” in *Web Intelligence*, no. 1 in 15, pp. 1–17, IOS Press, 2017.

[52] D. Alajbeg, Z. Bubaš, and Š. Ivan, “The p/e effect on the croatian stock market,” *Journal of International Scientific Publications: Economy and Business*, vol. 10, p. 84, 2016.

[53] K. Schipper and A. Smith, “A comparison of equity carve-outs and seasoned equity offerings: Share price effects and corporate restructuring,” *Journal of Financial Economics*, vol. 15, no. 1-2, pp. 153–186, 1986.

[54] R. M. Hayes, “The impact of trading commission incentives on analysts’ stock coverage decisions and earnings forecasts,” *Journal of Accounting Research*, vol. 36, no. 2, pp. 299–320, 1998.

[55] K.-i. Kamijo and T. Tanigawa, “Stock price pattern recognition-a recurrent neural network approach,” in *1990 IJCNN International Joint Conference on Neural Networks*, pp. 215–221, IEEE, 1990.

[56] C. H. Lee and K. C. Park, “Prediction of monthly transition of the composition stock price index using recurrent back-propagation,” in *Artificial neural networks*, pp. 1629–1632, Elsevier, 1992.

[57] E. Guresen, G. Kayakutlu, and T. U. Daim, “Using artificial neural network models in stock market index prediction,” *Expert Systems with Applications*, vol. 38, no. 8, pp. 10389–10397, 2011.

[58] S. Gaoand M. Zhouand Y. Wangand J. Chengand H. Yachiand J. Wang, “Dendritic neuron model with effective learning algorithms for classification, approximation, and prediction,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, pp. 601–614, Feb 2019.

[59] D. Jiaand S. Zhengand L. Yangand Y. Todoand S. Gao, “A dendritic neuron model with nonlinearity validation on istanbul stock and taiwan futures exchange indexes prediction,” in *2018 5th IEEE International Conference on Cloud Computing and Intelligence Systems (CCIS)*, pp. 242–246, Nov 2018.

[60] T. Zhou, S. Gao, J. Wang, C. Chu, Y. Todo, and Z. Tang, “Financial time series prediction using a dendritic neuron model,” *Knowledge-Based Systems*, vol. 105, pp. 214 – 224, 2016.

[61] J. D. Farmer and A. W. Lo, “Frontiers of finance: Evolution and efficient markets,” *Proceedings of the National Academy of Sciences*, vol. 96, no. 18, pp. 9991–9992, 1999.

[62] V. S. Pagolu, K. N. Reddy, G. Panda, and B. Majhi, “Sentiment analysis of twitter data for predicting stock market movements,” in *2016 international conference on signal processing, communication, power and embedded system (SCOPES)*, pp. 1345–1350, IEEE, 2016.

[63] A. Mittal and A. Goel, “Stock prediction using twitter sentiment analysis,” *Standford University, CS229 (2011 http://cs229. stanford. edu/proj2011/GoelMittal-StockMarketPredictionUsingTwitterSentimentAnalysis. pdf)*, vol. 15, 2012.

[64] N. Oliveira, P. Cortez, and N. Areal, “The impact of microblogging data for stock market prediction: Using twitter to predict returns, volatility, trading volume and survey sentiment indices,” *Expert Systems with Applications*, vol. 73, pp. 125–144, 2017.

[65] S. Consoli, D. Reforgiato Recupero, and M. Saisana, “Data science for economics and finance: Methodologies and applications,” 2021.

[66] T. B. Trafalis and H. Ince, “Support vector machine for regression and applications to financial forecasting,” in *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, vol. 6, pp. 348–353, IEEE, 2000.

[67] C. Cortes and V. Vapnik, “Support-vector networks,” *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[68] B. M. Henrique, V. A. Sobreiro, and H. Kimura, “Literature review: Machine learning techniques applied to financial market prediction,” *Expert Systems with Applications*, 2019.

[69] C. Tsai and S. Wang, “Stock price forecasting by hybrid machine learning techniques,” in *Proceedings of the International MultiConference of Engineers and Computer Scientists*, vol. 1, p. 60, 2009.

[70] D. Shah, Isah, and F. Zulkernine, “Stock market analysis: A review and taxonomy of prediction techniques,” *International Journal of Financial Studies*, vol. 7, p. 26, 05 2019.

[71] M. Ballings, D. V. den Poel, N. Hespeels, and R. Gryp, “Evaluating multiple classifiers for stock price direction prediction,” *Expert Systems with Applications*, vol. 42, no. 20, pp. 7046 – 7056, 2015.

[72] S. Basak, S. Kar, S. Saha, L. Khaidem, and S. Dey, “Predicting the direction of stock market prices using tree-based classifiers,” *The North American Journal of Economics and Finance*, 07 2018.

[73] S. Dey, Y. Kumar, S. Saha, and S. Basak, “Forecasting to classification: Predicting the direction of stock market price using xtreme gradient boosting,” 10 2016.

[74] S. Aggarwal, L. Saini, and A. Kumar, “Price forecasting using wavelet transform and lse based mixed model in australian electricity market,” *International Journal of Energy Sector Management*, vol. 2, pp. 521–546, 11 2008.

[75] Y. Wuand J. Maoand W. Li, “Predication of futures market by using boosting algorithm,” in *International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*, pp. 1–4, March 2018.

[76] S. M. Idreesand M. A. Alamand P. Agarwal, “A prediction approach for stock market volatility based on time series data,” *IEEE Access*, vol. 7, pp. 17287–17298, 2019.

[77] S. Carta, A. Medda, A. Pili, D. Reforgiato Recupero, and R. Saia, "Forecasting e-commerce products prices by combining an autoregressive integrated moving average (arima) model and google trends data," *Future Internet*, vol. 11, no. 1, p. 5, 2019.

[78] H. P. S. D. Weerathungaand A. T. P. Silva, "Drnn-arima approach to short-term trend forecasting in forex market," in *International Conference on Advances in ICT for Emerging Regions (ICTer)*, pp. 287–293, Sep. 2018.

[79] J. Chouand T. Nguyen, "Forward forecast of stock price using sliding-window metaheuristic-optimized machine-learning regression," *IEEE Transactions on Industrial Informatics*, vol. 14, pp. 3132–3142, July 2018.

[80] J. Nobre and R. F. Neves, "Combining principal component analysis, discrete wavelet transform and xgboost to trade in the financial markets," *Expert Systems with Applications*, vol. 125, pp. 181 – 194, 2019.

[81] D. Gupta, M. Pratama, Z. Ma, J. Li, and M. Prasad, "Financial time series forecasting using twin support vector regression," *PLOS ONE*, vol. 14, pp. 1–27, 03 2019.

[82] M. Prasad, Y. Lin, C. Lin, M. Er, and O. Prasad, "A new data-driven neural fuzzy system with collaborative fuzzy clustering mechanism," *Neurocomputing*, vol. 167, pp. 558 – 568, 2015.

[83] O. P. Pateland N. Bharilland A. Tiwariand M. Prasad, "A novel quantum-inspired fuzzy based neural network for data classification," *IEEE Transactions on Emerging Topics in Computing*, pp. 1–1, 2019.

[84] G. J. Klir and T. A. Folger, *Fuzzy Sets, Uncertainty, and Information.* Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1987.

[85] R. Gonçalves, V. M. Ribeiro, F. L. Pereira, and A. P. Rocha, "Deep learning in exchange markets," *Information Economics and Policy*, vol. 47, pp. 38 – 51, 2019. The Economics of Artificial Intelligence and Machine Learning.

[86] S. P. Chatzis, V. Siakoulis, A. Petropoulos, E. Stavroulakis, and N. Vlachogiannakis, "Forecasting stock market crisis events using deep and statistical machine learning techniques," *Expert Systems with Applications*, vol. 112, pp. 353 – 371, 2018.

[87] Y. Dengand F. Baoand Y. Kongand Z. Renand Q. Dai, "Deep direct reinforcement learning for financial signal representation and trading,"

IEEE Transactions on Neural Networks and Learning Systems, vol. 28, pp. 653–664, March 2017.

[88] T. G. Dietterich, "Ensemble methods in machine learning," in *Multiple Classifier Systems*, vol. 1857 of *Lecture Notes in Computer Science*, pp. 1–15, Springer, 2000.

[89] A. Zainal, M. A. Maarof, S. M. H. Shamsuddin, and A. Abraham, "Ensemble of one-class classifiers for network intrusion detection system," in *IAS*, pp. 180–185, IEEEComputer Society, 2008.

[90] T. G. Dietterich, "Ensemble methods in machine learning," in *International workshop on multiple classifier systems*, pp. 1–15, Springer, 2000.

[91] R. Saia, C. Salvatore, and R. RECUPERO, "A probabilistic-driven ensemble approach to perform event classification in intrusion detection system," in *10th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*, 2018.

[92] S. Carta, G. Fenu, D. R. Recupero, and R. Saia, "Fraud detection for e-commerce transactions by employing a prudential multiple consensus model," *Journal of Information Security and Applications*, vol. 46, pp. 13–22, 2019.

[93] O. Sagi and L. Rokach, "Ensemble learning: Asurvey," *Wiley Interdisc. Rew.: Data Mining and Knowledge Discovery*, vol. 8, no. 4, 2018.

[94] B. Zhu, S. Ye, P. Wang, K. He, T. Zhang, and Y.-M. Wei, "A novel multiscale nonlinear ensemble leaning paradigm for carbon price forecasting," *Energy Economics*, vol. 70, pp. 143 – 157, 2018.

[95] A. P. Rattoand S. Merelloand L. Onetoand Y. Maand L. Malandriand E. Cambria, "Ensemble of technical analysis and machine learning for market trend prediction," in *IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 2090–2096, Nov 2018.

[96] S. Sun, Y. Sun, S. Wang, and Y. Wei, "Interval decomposition ensemble approach for crude oil price forecasting," *Energy Economics*, vol. 76, pp. 274 – 287, 2018.

[97] K. S. Ganand K. O. Chinand P. Anthonyand S. V. Chang, "Homogeneous ensemble feedforward neural network in cimb stock price forecasting," in *International Conference on Artificial Intelligence in Engineering and Technology (IICAIET)*, pp. 1–6, Nov 2018.

[98] Y. Ding, “A novel decompose-ensemble methodology with aic-ann approach for crude oil forecasting,” *Energy*, vol. 154, pp. 328 – 336, 2018.

[99] T. Kimoto, K. Asakawa, M. Yoda, and M. Takeoka, “Stock market prediction system with modular neural networks,” in *1990 IJCNN international joint conference on neural networks*, pp. 1–6, IEEE, 1990.

[100] K.-i. Kamijo and T. Tanigawa, “Stock price pattern recognition-a recurrent neural network approach,” in *1990 IJCNN International Joint Conference on Neural Networks*, pp. 215–221, IEEE, 1990.

[101] C. H. Lee and K. C. Park, “Prediction of monthly transition of the composition stock price index using recurrent back-propagation,” in *Artificial neural networks*, pp. 1629–1632, Elsevier, 1992.

[102] T. Sunand J. Wangand J. Niand Y. Caoand B. Liu, “Predicting futures market movement using deep neural networks,” in *18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, pp. 118–125, 2019.

[103] Y. Linand T. Huangand W. Chungand Y. Ueng, “Forecasting fluctuations in the financial index using a recurrent neural network based on price features,” *IEEE Transactions on Emerging Topics in Computational Intelligence*, pp. 1–12, 2020.

[104] W. Fenghua, X. Jihong, H. Zhifang, and G. Xu, “Stock price prediction based on ssa and svm,” *Procedia Computer Science*, vol. 31, pp. 625 – 631, 2014. 2nd International Conference on Information Technology and Quantitative Management, ITQM 2014.

[105] Z. Zhou, M. Gao, Q. Liu, and H. Xiao, “Forecasting stock price movements with multiple data sources: Evidence from stock market in china,” *Physica A: Statistical Mechanics and its Applications*, vol. 542, p. 123389, 2020.

[106] Z. Tan, Z. Yan, and G. Zhu, “Stock selection with random forest: An exploitation of excess return in the chinese stock market,” *Heliyon*, vol. 5, no. 8, p. e02310, 2019.

[107] W. Khan, M. A. Ghazanfar, M. A. Azam, A. Karami, K. H. Alyoubi, and A. S. Alfakeeh, “Stock market prediction using machine learning classifiers and social media, news,” *Journal of Ambient Intelligence and Humanized Computing*, 2020.

[108] Z. Wang and T. Oates, “Encoding time series as images for visual inspection and classification using tiled convolutional neural networks,”

in *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[109] O. Mihatsch and R. Neuneier, “Risk-sensitive reinforcement learning,” in *Advances in Neural Information Processing Systems*, pp. 1031–1037, MIT Press, 1999.

[110] X. Gao, S. Hongkong, and L. Chan, “An algorithm for trading and portfolio management using q-learning and sharpe ratio maximization,” in *International Conference on Neural Information Processing*, pp. 832–837, 2000.

[111] J. W. Lee, J. Park, O. Jangmin, J. Lee, and E. Hong, “A multiagent approach to \$q\$-learning for daily stock trading,” *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 37, pp. 864–877, 2007.

[112] J. Moody, L. Wu, Y. Liao, and M. Saffell, “Performance functions and reinforcement learning for trading systems and portfolios,” *Journal of Forecasting*, vol. 17, no. 5-6, pp. 441–470, 1998.

[113] Q. Kang, H. Zhou, and Y. Kang, “An asynchronous advantage actor-critic reinforcement learning method for stock selection and portfolio management,” in *Proceedings of the 2nd International Conference on Big Data Research*, ICBDR 2018, (New York, NY, USA), p. 141–145, Association for Computing Machinery, 2018.

[114] K. Lei, B. Zhang, Y. Li, M. Yang, and Y. Shen, “Time-driven feature-aware jointly deep reinforcement learning for financial signal representation and algorithmic trading,” *Expert Systems with Applications*, vol. 140, p. 112872, 2020.

[115] S. Choi, “Independent component analysis,” *Encyclopedia of Biometrics*, pp. 917–924, 2015.

[116] A. Hyvärinen and E. Oja, “Independent component analysis: algorithms and applications,” *Neural networks*, vol. 13, no. 4-5, pp. 411–430, 2000.

[117] C. Jutten and J. Karhunen, “Advances in blind source separation (bss) and independent component analysis (ica) for nonlinear mixtures,” *International Journal of Neural Systems*, vol. 14, no. 05, pp. 267–292, 2004.

[118] C. Jutten and J. Herault, “Blind separation of sources, part i: An adaptive algorithm based on neuromimetic architecture,” *Signal processing*, vol. 24, no. 1, pp. 1–10, 1991.

[119] W. Huang, Y. Nakamori, and S.-Y. Wang, "Forecasting stock market movement direction with support vector machine," *Computers & Operations Research*, vol. 32, no. 10, pp. 2513–2522, 2005.

[120] J. Döpke, U. Fritsche, and C. Pierdzioch, "Predicting recessions with boosted regression trees," *International Journal of Forecasting*, vol. 33, no. 4, pp. 745–759, 2017.

[121] C. D. Kirkpatrick and J. R. Dahlquist, *Technical Analysis: The Complete Resource for Financial Market Technicians.* FT Press, November 2010.

[122] E. Tomasini and U. Jaekle, *Trading Systems.* Harriman House Limited, 2011.

[123] A. J. C. SHARKEY, "On combining artificial neural nets," *Connection Science*, vol. 8, no. 3-4, pp. 299–314, 1996.

[124] A. Tsymbal, M. Pechenizkiy, and P. Cunningham, "Diversity in search strategies for ensemble feature selection," *Information fusion*, vol. 6, no. 1, pp. 83–98, 2005.

[125] M. Van Wezel and R. Potharst, "Improved customer choice predictions using ensemble methods," *European Journal of Operational Research*, vol. 181, no. 1, pp. 436–452, 2007.

[126] D. Enke, M. Grauer, and N. Mehdiyev, "Stock market prediction with multiple regression, fuzzy type-2 clustering and neural networks," *Procedia Computer Science*, vol. 6, pp. 201–206, 2011.

[127] M. Klassen, "Investigation of some technical indexes in stock forecasting using neural networks.," in *WEC (5)*, pp. 75–79, Citeseer, 2005.

[128] P. C. Tetlock, "Giving content to investor sentiment: The role of media in the stock market," *The Journal of finance*, vol. 62, no. 3, pp. 1139–1168, 2007.

[129] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[130] I. A. T. Hashem, N. B. Anuar, A. Gani, I. Yaqoob, F. Xia, and S. U. Khan, "Mapreduce: Review and open challenges," *Scientometrics*, vol. 109, no. 1, pp. 389–422, 2016.

[131] S. Barra, S. M. Carta, A. Corriga, A. S. Podda, and D. Reforgiato Recupero, "Deep learning and time series-to-image encoding for financial forecasting," *IEEE/CAA Journal of Automatica Sinica*, vol. 7, no. JAS-2019-0392, p. 683, 2020.

[132] Z. Wang and T. Oates, "Encoding time series as images for visual inspection and classification using tiled convolutional neural networks," in *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[133] Z. Wang and T. Oates, "Imaging time-series to improve classification and imputation," in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[134] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[135] K. Heand X. Zhangand S. Renand J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, June 2016.

[136] P. Cizeau, Y. Liu, M. Meyer, C.-K. Peng, and H. E. Stanley, "Volatility distribution in the s&p500 stock index," *Physica A: Statistical Mechanics and its Applications*, vol. 245, no. 3-4, pp. 441–445, 1997.

[137] M. Martens, "Measuring and forecasting s&p 500 index-futures volatility using high-frequency data," *Journal of Futures Markets: Futures, Options, and Other Derivative Products*, vol. 22, no. 6, pp. 497–518, 2002.

[138] D. H. Wolpert, "Stacked generalization," *Neural Networks*, vol. 5, pp. 241–259, 1992.

[139] H. V. Hasselt, "Double q-learning," in *Advances in Neural Information Processing Systems 23* (J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, eds.), pp. 2613–2621, Curran Associates, Inc., 2010.

[140] S. Barraand S. M. Cartaand A. Corrigaand A. S. Poddaand D. R. Recupero, "Deep learning and time series-to-image encoding for financial forecasting," *IEEE/CAA Journal of Automatica Sinica*, vol. 7, no. 3, pp. 683–692, 2020.

[141] H. S. Sim, H. I. Kim, and J. J. Ahn, "Is deep learning for image recognition applicable to stock market prediction?," *Complexity*, 2019.

[142] T. Kim and H. Y. Kim, "Forecasting stock prices with a feature fusion lstm-cnn model using different representations of the same data," *PLOS ONE*, vol. 14, pp. 1–23, 02 2019.

[143] J. Han, M. Kamber, and J. Pei, *Data Transformation and Data Discretization*, ch. 3, pp. 111–118. Elsevier, 2011.

[144] R. Surton and A. Barto, *Reinforcement Learning: an introduction*, vol. 1. MIT press Cambridge, 1998.

[145] M. Puterman, *Markov decision processes: discrete stochastic dynamic programming.* John Wiley & Sons, 2014.

[146] S. M. J. Jalali, S. Ahmadian, A. Khosravi, S. Mirjalili, M. R. Mahmoudi, and S. Nahavandi, "Neuroevolution-based autonomous robot navigation: A comparative study," *Cognitive Systems Research*, vol. 62, pp. 35 – 43, 2020.

[147] W. Siand J. Liand P. Dingand R. Rao, "A multi-objective deep reinforcement learning approach for stock index future's intraday trading," in *International Symposium on Computational Intelligence and Design (ISCID)*, vol. 2, pp. 431–436, Dec 2017.

[148] H. V. Hasselt, "Double q-learning," in *Advances in Neural Information Processing Systems 23* (J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, eds.), pp. 2613–2621, Curran Associates, Inc., 2010.

[149] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *International Conference on Machine Learning*, vol. 48 of *Proceedings of Machine Learning Research*, (New York, New York, USA), pp. 1995–2003, PMLR, 2016.

[150] O. B. Sezer and A. M. Ozbayoglu, "Algorithmic financial trading with deep convolutional neural networks: Time series to image conversion approach," *Applied Soft Computing*, vol. 70, pp. 525 – 538, 2018.

[151] G. G. Calviand V. Lucicand D. P. Mandic, "Support tensor machine for financial forecasting," in *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 8152–8156, May 2019.

[152] M. Plappert, "keras-rl." `https://github.com/keras-rl/keras-rl`, 2016.

[153] A. W. Lo, "The statistics of sharpe ratios," *Financial analysts journal*, vol. 58, no. 4, pp. 36–52, 2002.

[154] A. Chaudhry and H. L. Johnson, "The efficacy of the sortino ratio and other benchmarked performance measures under skewed return distributions," *Australian Journal of Management*, vol. 32, no. 3, pp. 485–502, 2008.

[155] M. Magdon-Ismail and A. F. Atiya, "Maximum drawdown," *Risk Magazine*, vol. 17, no. 10, pp. 99–102, 2004.

[156] S. Ahmadianand A. R. Khanteymoori, "Training back propagation neural networks using asexual reproduction optimization," in *Conference on Information and Knowledge Technology (IKT)*, pp. 1–6, 2015.

[157] S. M. J. Jalali, S. Ahmadian, P. M. Kebria, A. Khosravi, C. P. Lim, and S. Nahavandi, "Evolving artificial neural networks using butterfly optimization algorithm for data classification," in *Neural Information Processing*, (Cham), pp. 596–607, Springer International Publishing, 2019.