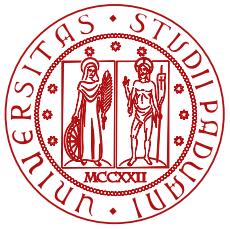


Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Master Thesis in Computer Engineering - Artificial Intelligence and Robotics

Stock Market Forecast through Neural Networks

Master Candidate

Tommaso Bianco

Student ID 2037864

Supervisor

Prof. Loris Nanni

University of Padova

Co-supervisor

Dott. Riccardo De Socio

University of Padua

Academic Year
2023/2024
Graduation Date
23rd October 2024

Abstract

This document presents the findings of an investigation into the potential applications of deep learning techniques, specifically Long Short-Term Memory (LSTM) networks and hybrid LSTM-Convolutional Neural Networks (CNNs), in the forecasting of closing prices of Exchange-Traded Funds (ETFs) and Stocks. The study of the market presents a significant complexity in financial forecasting, as well as a considerable challenge in accurately predicting short-term price movements. This research is concerned with the construction of models that integrate a variety of financial data as input, including historical closing prices, macroeconomic indicators and the transformation of the input data, such as wavelets and time cycles features. Furthermore, the study investigates the potential of Graph Convolutional Networks (GCNs) and the potential of the hybrid model of GCN and LSTM, which may be capable of capturing short-term trends and the relational dependencies between different financial variables. Despite the difficulties inherent in financial prediction, the models developed demonstrate promising results, although they face challenges in precise timing. The findings indicate the advent of the use of neural networks in financial forecasting for future real-world applications.

Contents

List of Figures

List of Tables

1

Introduction

Exchange-Traded Funds (ETFs) have become increasingly popular investments due to their diversification benefits, liquidity and cost-effectiveness. ETFs follow different paths: sectors, indices or commodities and so predicting their performance is therefore of great interest to both individual investors and financial institutions. This thesis involves the study of the methods for predicting ETF values by learning the historical closing price movements, the Stock price movements and also a list of macroeconomic indicators.

The so-called Artificial Intelligence (AI) and in particular the Deep Learning, in these years of continuous development, have become significantly important in many fields and can provide numerous tools for data analysis and predictive modelling. Since Deep Learning models have the ability to learn from large amounts of data and generalise them across different scenarios, these technologies excel in identifying intricate patterns and relationships within large datasets, making them well suited for the complexities of the financial market and improving the accuracy of financial predictions.

This paper uses advanced AI approaches to predict the future trends of the ETFs and the Stock Markets by analysing various key input features such as ETF closing prices, stock prices and macroeconomic indicators. By constructing predictive models that integrate these different data sources, the aim is to give a first impression on how to handle the ETF forecast. The aim of this approach is not only to help improve investment strategies, but also to contribute to the study of how the various economic variables can interact to determine the performance of ETFs.

The research focused on the use of neural networks, in particular Long Short-Term Memory (LSTM) networks and hybrid models combining LSTM with Convolutional Neural Networks (CNNs), to predict the closing prices of ETFs and Stocks. The objective is to evaluate how well these models perform against a traditional baseline model and to assess their ability to capture short-term market trends.

Another important contribution to this thesis, which will be subject to further improvement in the future, is the exploration of Graph Convolutional Networks (GCNs). These have the potential to enhance performance and provide a new perspective on the interactions between financial variables. This approach can improve the predictions by taking into account the structural relationships between different financial instruments. The structure of this thesis is as follows: Chapter 2 outlines the methodology used, including data pre-processing and the implementation of deep learning models. Chapter 3 presents the results of the LSTM and LSTM-Convolutional models, while Chapter 4 concludes with a discussion of the findings and suggestions for future work.

2

Methods

This chapter presents the methods used to achieve the objective of the thesis: to forecast Stock market closing prices.
Given the complexity and the volume of the data involved, a robust and systematic approach is essential. This chapter contains the different stages of the methodology, from data collection and preprocessing to the implementation of machine learning models and finally a chapter where there is an evaluation of their performance.
It begins with a discussion of the data collection process, highlighting the sources of stock and macroeconomic data and the steps taken to ensure data integrity and consistency. The preprocessing phase involves the cleaning of the data and the performing of feature engineering in order to extract relevant features that can improve the performance of the model.
The core of this chapter is the methodology: it is about the application of deep learning techniques. In this study, there is the exploration of various machine learning algorithms, including convolutional networks, lstm networks and a mixture of both.
Throughout this study, the methods are all executed using Python as the primary programming language. Python's extensive ecosystem of libraries and tools for data analysis and machine learning, such as Pandas, Numpy, scikit-learn, TensorFlow and Pytorch, provided a flexible framework for the research. These libraries facilitated efficient data manipulation, model development, and evaluation, allowing to focus on refining our predictive models and interpreting their results. At the end of this chapter, the reader will have a thorough under-

standing of the computational techniques and tools used in this study and how they contribute to the goal of predicting ETF and Stock market values.

2.1 Data

The Data used can be divided into several categories: ETF tickers, Stocks, Macro-Economics indices.
The accuracy and reliability of any forecasting model is highly dependent on the quality and relevance of the data used. The data for this study is sourced from a variety of sources and sites in order to provide a view of this financial market.

ETF

ETFs are investment funds or SICAVs with low management fees that are traded on the stock exchange in the same manner as ordinary shares. ETFs represent a specific type of investment fund that combines the most advantageous characteristics of two distinct asset classes: they offer the diversification benefits typically associated with mutual funds while providing the convenience and accessibility associated with stocks.
An ETF is a basket of investments such as stocks or bonds. ETFs permit the investment in numerous securities simultaneously, and ETFs frequently have lower fees than other types of funds[11].
The historical closing price data for ETFs was employed, covering various sectors and indices. This includes information on ETF composition and performance metrics. The following table provides a summary of the ETFs utilized, accompanied by the corresponding sector designation.
This data are taken from the Yahoo Finance data service, which contains all the closing prices of the major ETFs since they was created.

ETF	SECTOR
XLB	Materials
XLE	Energy
XLF	Financials
XLI	Industrials
XLK	Information Technology
XLP	Consumer Staples
XLU	Utilities
XLV	Healt Care
XLY	Consumer Discretionary

Exchange-traded funds work like this: The fund provider owns the underlying assets, it designs a fund to keep tracking their performance and then sells shares in that fund to investors. Shareholders don't own the underlying assets in the fund, they only own a portion of an ETF. Even so, investors in an ETF that tracks a stock index may get dividend payments for any dividend stocks in the index.

Stock Market Data

A stock, also known as equity, is a security that represents the ownership of a fraction of the issuing corporation. Units of stock are called shares, which entitle the owner to a proportion of the corporation's assets and profits equal to how much stock they own[1].
Stocks are bought and sold predominantly on stock exchanges and have to conform to government regulations meant to protect investors from fraudulent practices.
There were used the historical closing price data for a number equal to 386 Stocks in order to diversificate the market and have data that can be connected to the ETF tickers used. All this data are taken from Yahoo Finance and constitute an important help for the models.

Macroeconomic Indicators

Macroeconomic indicators are statistics or data readings that reflect the economic circumstances of a particular country, region or sector. They are used by analysts and governments to assess the current and future health of the economy and financial markets[3].

In this case key economic indicators were sourced from governmental databases, central banks, and international financial organizations. For the thesis there were used only the following leading indicators: Money Supply M2, Fed Balance Sheet (in trillions), Producer Price Index (PPI) Manufacturing, Producer Price Index (PPI) Core, Purchasing Managers' Index (PMI) Manufacturing, Purchasing Managers' Index (PMI) Services, Michigan Consumer Sentiment, OECD Consumer Confidence, Case-Shiller Home Prices, New Houses Sold.

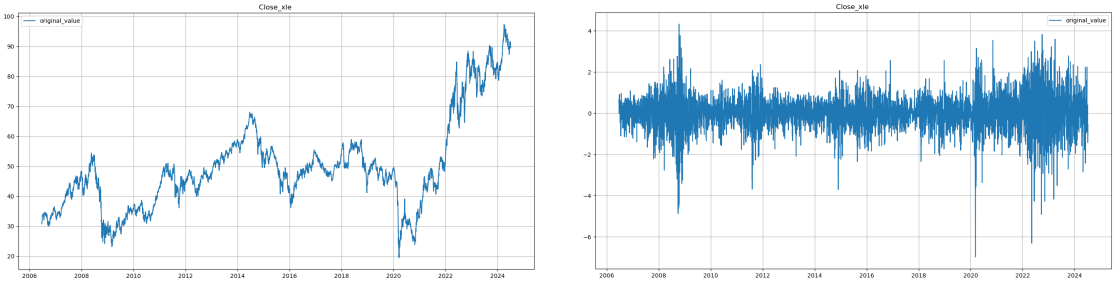
The important steps to be followed to have the input tensors of the models were divided into: Data collection, Data Preprocessing and Feature Engineering.

2.1.1 Data Collection

For Data collection, an automated script was written in Python that downloads and aggregates data from the aforementioned sources. APIs used is *yfinance* from Yahoo Finance that ensure efficient and timely data retrieval. The collected data spans a period of twenty years to capture a wide range of market conditions and economic cycles.

2.1.2 Data Preprocessing

Preprocessing is a critical step for the methodology, it is essential to convert raw data into a suitable format for analysis. All the data have been divided into different data frames indexed by day.
Depending on the deep learning model to use, if it was needed, ETF closing prices data and Stocks data were transformed by calculating the difference between each value and its preceding value in order to calculate the derivative, subsequently, any Nan values are replaced with zero to ensure a complete dataset. This transformation was done in order to uniform the data from different ETF tickers and Stocks, because they have different closing prices from each other. By using the derivative function, that is the *shift* function of Pandas, data becomes also negative. Here there are two examples of how data were managed: in Figure 2.1a there is a plot of the closing price of the ticker XLE, in Figure 2.1b there is a plot of the closing price of the ticker XLE with the subtraction of the shifted data.



(a) Closed prices of the ETF ticker XLE (b) Closed prices with subtracted shift

Figure 2.1: Comparison of the closed prices and shifted values for the ETF ticker XLE

Then, ETF and Stocks data were transformed using different Scaler from *sklearn*[1]. Several tests have been done in order to choose the correct transformer function to be used for the model.
First one was *PowerTransformer* with Box-Cox method. *PowerTransformer* Apply a power transform featurewise to make data more Gaussian-like. The Box-Cox method of this transformer uses a parameter to apply to the data:

$$y(\lambda) = \begin{cases} \frac{y^\lambda - 1}{\lambda} & \text{if } \lambda \neq 0 \\ \log(y) & \text{if } \lambda = 0 \end{cases}$$

and it can be applied only to positive data. If the data were preprocessing by using the derivative function explained before, this method was modified with the differentiation from positive values and negative values. This version requires that the positive data are transformed directly with the Box-Cox function, while the negative data are first transformed into positive, then the Box-Cox function is applied to them, and then transformed back into negative.
However, the models did not produce satisfactory results with the Box-Cox transformation, necessitating the use of the Yeo-Johnson transformation instead. The Yeo-Johnson transformation can handle both positive and negative values and is defined as follows:

[1] `scikit-learn` is a Python library for machine learning that provides simple and efficient tools for data analysis and modeling. For more information, visit `https://scikit-learn.org`

$$y(\lambda) = \begin{cases} \frac{(y+1)^{\lambda}-1}{\lambda} & \text{if } \lambda \neq 0 \text{ and } y \geq 0 \\ \log(y+1) & \text{if } \lambda = 0 \text{ and } y \geq 0 \\ -\frac{(-y+1)^{2-\lambda}-1}{2-\lambda} & \text{if } \lambda \neq 2 \text{ and } y < 0 \\ -\log(-y+1) & \text{if } \lambda = 2 \text{ and } y < 0 \end{cases}$$

The Yeo-Johnson is a power transform family of functions applied to create a monotonic transformation of data using power functions. It is a technique used to stabilize variance, make the data more normal distribution-like, improve the validity of measures of association (such as the Pearson correlation between variables), and for other data stabilization procedures. PowerScaling is crucial for making data more Gaussian, which can improve model performance, particularly in neural networks. Howerver, as the data values approach the maximun value, the gradient can explode, leading to instability and causing the gradient to go to infinity. This phenomenon is often referred to as the exploding gradient problem.
The MinMaxScaler in Scikit-Learn is a preprocessing technique used to rescale the features of a dataset so that they fall within a specified range that is [0, 1]. This scaling is achieved by transforming each feature individually according to the following formula:

$$X' = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}}$$

where X is the original feature value, X_{min} and X_{max} are the minimum and maximum values of the feature, respectively, X' is the scaled feature value. This transformation ensures that all features have the same scale, which can improve the performance and convergence rate of many machine learning algorithms.
This transformation is often used as an alternative to zero mean, unit variance scaling. MinMaxScaler does not reduce the effect of outliers, but it linearly scales them down into a fixed range, where the largest occurring data point corresponds to the maximum value and the smallest one corresponds to the minimum value [7].
In figure 2.2 there is an example of distribution of the ETF ticker XLK. The distribution shown in the chart is clearly non-Gaussian, with a high concentration of values near zero and a long tail extending towards higher values. Accord-

ing to theory, using a power transformer could help in making the data more Gaussian, which would typically improve model performance. However, this research has shown that MinMax scaling was found to be more suitable, even though it doesn't normalize the data. MinMax scaling preserves the distribution range, which proved beneficial in this case, despite the absence of Gaussian-like normalization.

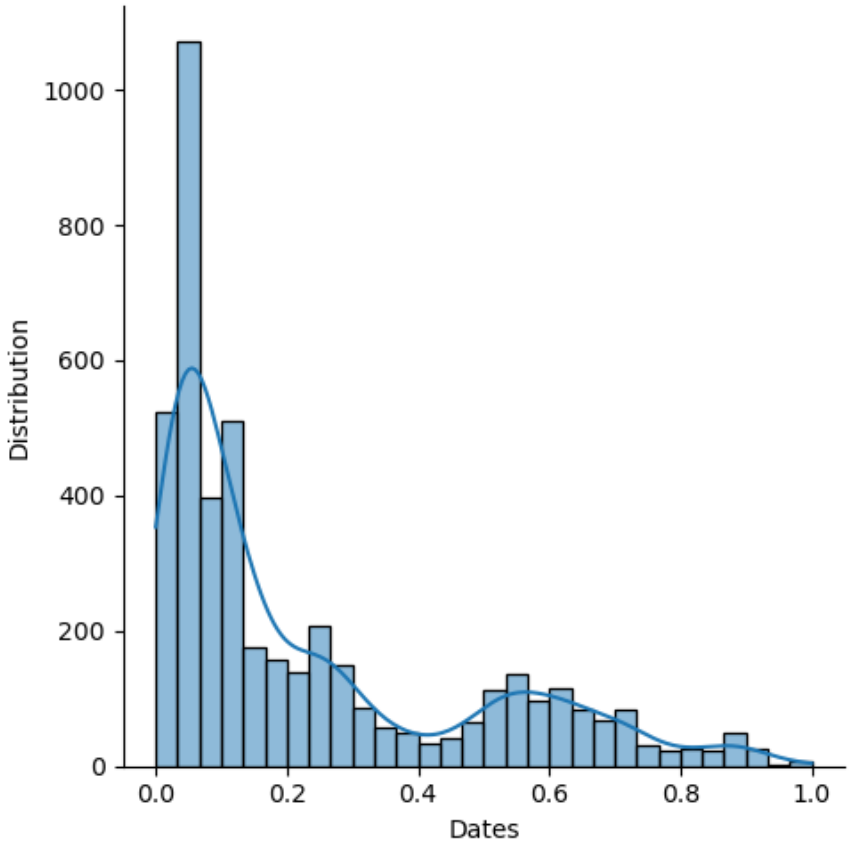


Figure 2.2: Distribution plot of the ticker XLK

The MinMax transformation was chosen for several reason: by scaling all features to the same range, it ensures that no single feature dominates the model due to its scale, scaling features to a common range can improve the performance and convergence rate of many machine learning algorithms and the nearly linear nature of MinMaxScaling preserves the distribution and relationships of the original data, making the transformed data more interpretable in the context of the original features. In Figure 2.3 there is an example of the closed price of the ticker XLE transformed by MinMaxScaling.

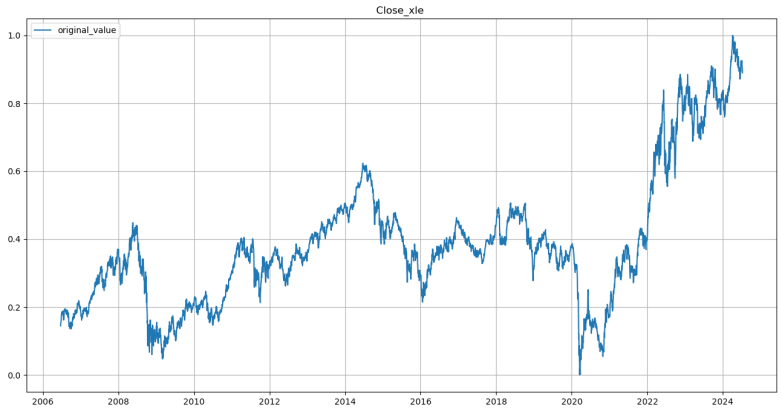


Figure 2.3: Closed prices of the ETF ticker XLE with the MinMaxScaler

As you can see from this Figure 2.3, the features of the dataset fall within the range [0, 1].
Those preprocessing methods were applied to the Stocks and ETF data, also the Macroeconomic Indicators were preprocessed by using the same methods.
The Macroeconomic Indicators were first preprocessed by subtracting the shifted value from the current value to create the derivative. Then, the values were transformed using the MinMaxScaler but with a different method: negative values were scaled to the range [-1, 0], while positive values were scaled to the range [0, 1]. This approach ensured that the entire range of macroeconomic data was represented consistently, facilitating better model training and performance.
All this data were inserted into 3D tensors in order to have them as input of the models. The ETF tickers and Stocks Market data were inserted into the same tensor following the indications that the model required: (samples, timesteps, features) or (timesteps, samples, features) where the features are the previous 20 days values for the input data closing price of the model and the subsequent 5 days values for the output data which was then compared to the model output data. Closing prices are widely used in financial analysis as they represent the final price at which an asset is traded during a trading session. When other values, such as open, high or low prices are not being investigated, the closing price serves as a key indicator of market and is often used to assess trends and make predictions.

2.1.3 Feature Engineering

In the data science and machine learning field, the process of feature engineering stands as a pivotal element in the journey from raw data to actionable insights. Feature engineering involves the creation, transformation and selection of features (variables) that can enhance the predictive power and performance of machine learning models.
For this thesis, the extracted meaningful features from raw datasets are then integrated with the original data to serve as input for the predictive models.

WAVELETS

A wavelet is a waveform of effectively limited duration that has a mean value of zero and a non-zero norm[8]. Wavelet transforms offer an advantage in that they can analyse the same signal in both the time and frequency domains, so in this case they are particularly useful for detecting non-stationary or time-varying characteristics. Wavelet transforms can perform the so called multi-resolution analyses, which allows the decomposition of the signals into several frequency bands at different scales, enhancing the important components while filtering out noise[2].
There are two types of wavelet transforms: continuous wavelet transform (CWT) and the discrete wavelet transform (DWT).
The CWT is a time-frequency transform, which is ideal for signals that have the frequency-domain representation that changes over time. These are similar to the short-time Fourier transform (STFT) with the difference that they use a variable-sized windows to tile the time-frequency plane.
The DWT scales are more coarse, making them more useful for compressing and denoising signals and images while preserving important features.
For the objective of this thesis the CWT was used. Stock market and ETF tickers data were analysed by this CWT in order to create a 3D tensor of this form: (samples, timesteps, features) or (timesteps, samples, features) where features are the numbers of logarithmic frequencies (30) into an interval from 10 to 100 day^{-1}. The type of wavelet used is *morl*. With this process it can be noticed that near the final values there is a problem called bottleneck in which wavelets try to look forward the data but do not find any data. In Figure 2.4 there is an example of CWT calculated on the XLI ticker.

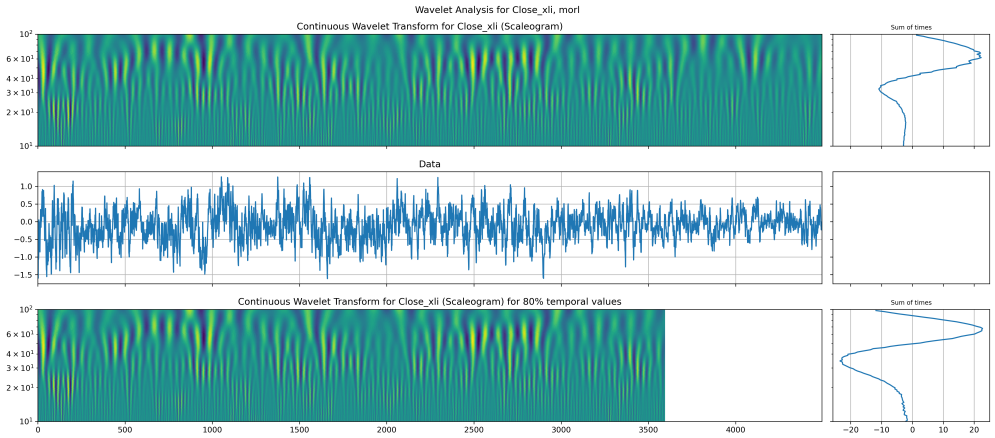


Figure 2.4: CWT calculated on the Closed price of XLI

To address the bottleneck issue, the same type of wavelet and the identical number of logarithm frequencies within the same interval were used. However, an algorithm was developed to create a 4D tensor that incorporates values from the 30 days preceding the current day[6]. Initially, the first 30 dates were initialized with their respective daily values. This approach significantly improved model performance.

TIME CYCLES

Knowing that the stock exchanges operate actively for 5 days a week, excluding Saturdays and Sundays, a transformation was implemented to convert linear day values into cyclical values based on both the day of the week and the day of the month.
To do this, in addition to excluding Saturdays and Sundays, holidays and extraordinary closing days were also managed. An algorithm was developed to take into account these closures, eliminating the resulting days, ensuring a correct historical series. The algorithm creates a modified calendar, it deletes manually those days and then calculates the final values with the formulas presented below.
For the day of the week (ranging from 1 to 5) the following formula was applied:

$$\frac{\sin\left(\frac{2\pi x}{\text{T\_days}} - \frac{\pi}{2}\right) + 1}{2}$$

Here x represents the day of the week and T_days is set to 5, reflecting the the

number of active days in a week.
similarly, for the day of the month (ranging from 1 to 30) the formula used was:

$$\frac{\sin\left(\frac{2\pi(x-1)}{\text{T\_month}} - \frac{\pi}{2}\right) + 1}{2}$$

In this context, x denotes the day of the month and T_month is set to 31 simplifying the assumption that each month consists of 31 days to ensure uniformity across different month lengths.
These transformations convert linear day representations into sinusoidal forms, capturing cyclical nature of weekly and monthly patterns in a manner conducive to machine learning model training and analysis. By introducing these cyclical features into the dataset, the model's ability to discern temporal dependencies was enhanced and the predictive accuracy across varying market conditions was improved.

ADJACENCY MATRIX

For the Graph Convolutional Network (GCN), one important feature is the Adjacency Matrix. For the purpose of this thesis, the Adjacency Matrix was created in order to establish connections between ETF tickers and Stocks based on their sector affiliations, it is a fundamental component in graph-based models, representing relationships or connections between nodes. In this study, a two-dimensional matrix was created where ETF tickers and Stocks were placed along both rows and columns. A value of 1 was assigned in the matrix where the ETF ticker and the Stock belonged to the same sector, indicating an adjacency or connectivity between them.
This approach leverages sector information to build a structured representation of relationships within the financial market ecosystem. By enhancing this sector-based associations into the Adjacency Matrix, the GCN can effectively capture sector-specific dependencies and interactions among different financial instruments. This representation ensures that the input data of Stocks and ETF tickers in the same sector can help between each other for the prediction of the model and also facilitates the model's ability to discern complex patterns and dependencies that are crucial for accurate predictive modeling and decision-making in financial markets.

2.2 Long-Short Term Memory (LSTM) Simple Model

In this section there is the description of the first model of the case of study: Long-Short Term Memory (LSTM) model.
It begins with a short description of the LSTM and then continues with the most interesting hyperparameters and with a description of all the relevant information of the model.

2.2.1 LSTM

Long-Short Term Memory (LSTM) is a particular type of recurrent neural network (RNN)[2] that has the aim of dealing with the vanishing gradient problem. It provides a short-term memory for RNN that can last thousands of timesteps, thus "long short-term memory".
In figure 2.5 there is the representation of the architecture of LSTM.

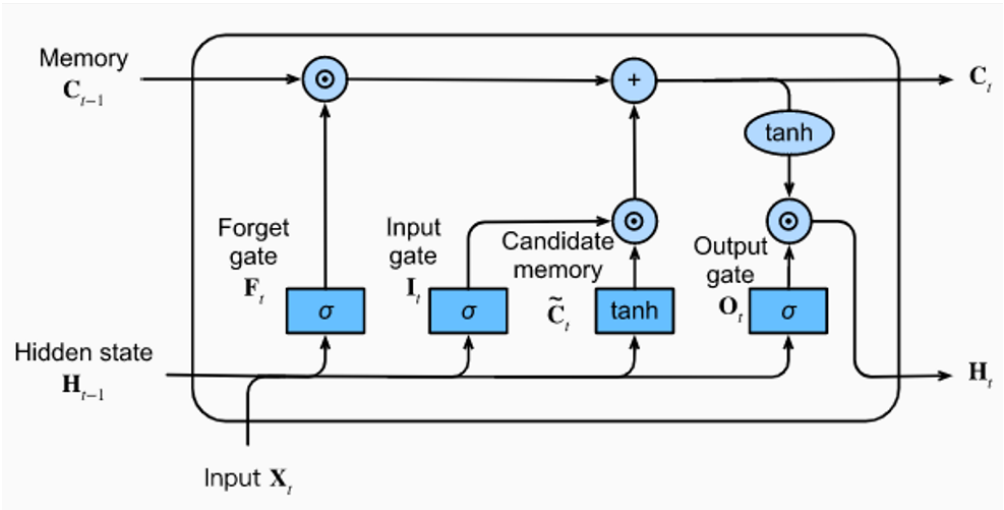


Figure 2.5: Architecture of LSTM

LSTM networks contains three gates:

1. Input gate I,
2. Output gate O,
3. Forget gate F.

[2]A recurrent neural network (RNN) is a type of artificial neural network that uses sequential data or time series data.

Each of these gates contains one layer with the sigmoid activation function. In addition there is a specialized single layer network $\overline{C}$ with a Tanh activation function.
There are also four state vectors:

1. Memory State C
2. Previous Memory State $C_{t\text{-}1}$
3. Hidden State H
4. Previous hidden State $H_{t\text{-}1}$

Given X_t as the input, $H_{t\text{-}1}$ with the learnable weights of the networks U,W,b and the candidate layer $\overline{C}_t$, the compact forms of the equations for the forward pass of an LSTM cell with a forget gate are the following:

$$F_t = \sigma(U_f * X_t + W_f * H_{t\text{-}1} + b_f)$$

$$I_t = \sigma(U_i * X_t + W_i * H_{t\text{-}1} + b_i)$$

$$O_t = \sigma(U_o * X_t + W_o * H_{t\text{-}1} + b_o)$$

and the output is:

$$H_t = O_t * tanh(C_t)$$

As it can be seen from the mathematical formulae, the forget gates determine which information is discarded from a previous state by assigning to that a value between 0 and 1 in comparison to the current input. A value near to 1 means to keep the information, a value near to 0 means to discard it. The input gate decides which pieces of new information to store in the current state, using the same system as forget gates. The output gate controls which pieces of information in the current state to output by assigning a value from 0 to 1 to the information, considering the previous and current states.
By using these gates, the LSTM network keep the relevant information from the current state and allows long-term dependencies to make predictions, both in current and future time-steps.

2.2.2 Architecture and Hyperparameters of the model

The first model used to address the thesis objective was a simple model that uses only the power of the LSTM, that is particularly suited for modeling time series data.
The goal of this model is to predict the values for the next 5 days based on the past closing prices of ETFs and stocks.
The model receives 3 different inputs:

- The input tensor that represents the historical closing prices of ETFs and stocks (input_1 in Figure 2.6).
- The input tensor that encapsulates the time cycles (input_2), this operation increases the vision of the model because it converts linear daily values into cyclical values.
- The input tensor that contains the macroeconomic data (input_3). For this 6 macroeconomic indicators were used.

Those input data are created with the form (samples,timesteps,features). This structure was used to ensure that the batch size is the samples, that are the different ETFs and Stocks market. With this implementation LSTMs are capable to capture patterns over time, so the model effectively processes sequential data and learns the dependencies between different timesteps.
The first operation performed on the tensor of the historical closing prices of ETFs and stocks is a one-dimensional convolutional layer, which applies 60 filters with a kernel size of 3 and a causal padding with the **tanh** activation function. The causal padding is a technique used in convolutional neural networks to ensure that the output at a specific time step depends only on past and present inputs, not future ones. It's an essential technique used for preventing the temporal order in sequential data, this ensures the model respects the causality of the data. This layer is designed to capture short-term dependencies in the time series data, extracting local features that can enhance the LSTM's ability to learn temporal patterns. With the 60 filters, the historical closing prices acquire much more value than the other inputs. The objective of this process was to balance the contribution of nodes coming from different inputs as they progressed deeper into the model. This allowed the model to harmonize diverse inputs and maintain consistent performance throughout its depth.
The result of this convolution is then concatenated with the other two tensors, creating a unified input for the subsequent layers. This concatenation allows

the model to simultaneously consider the different data sources, enriching the feature space available to the LSTM layers.
Following this concatenation, there are two stacked LSTM layers. The first LSTM layer consists of 128 units, configured to return sequences, which is necessary for the following LSTM layer that processes the entire sequence. The second LSTM layer has 64 units. These two layers are crucial for capturing the long-term dependencies in the data, enabling the model to learn from both recent and historical information.
The result of this two LSTM layers is passed through a TimeDistributed Dense layer with an output of 5 units, corresponding to the prediction of the next 5 days' values. This wrapper allows to apply a layer to every temporal slice of the input to generate a sequence of predictions rather than a single output.
This model architecture was chosen for its ability to handle complex temporal dependencies and to integrate multiple sources of data, providing a robust foundation for accurate time series forecasting.
The architecture of the model can be seen in the figure 2.6 and the visualization of the model by visualkeras can be seen in figure 2.7.

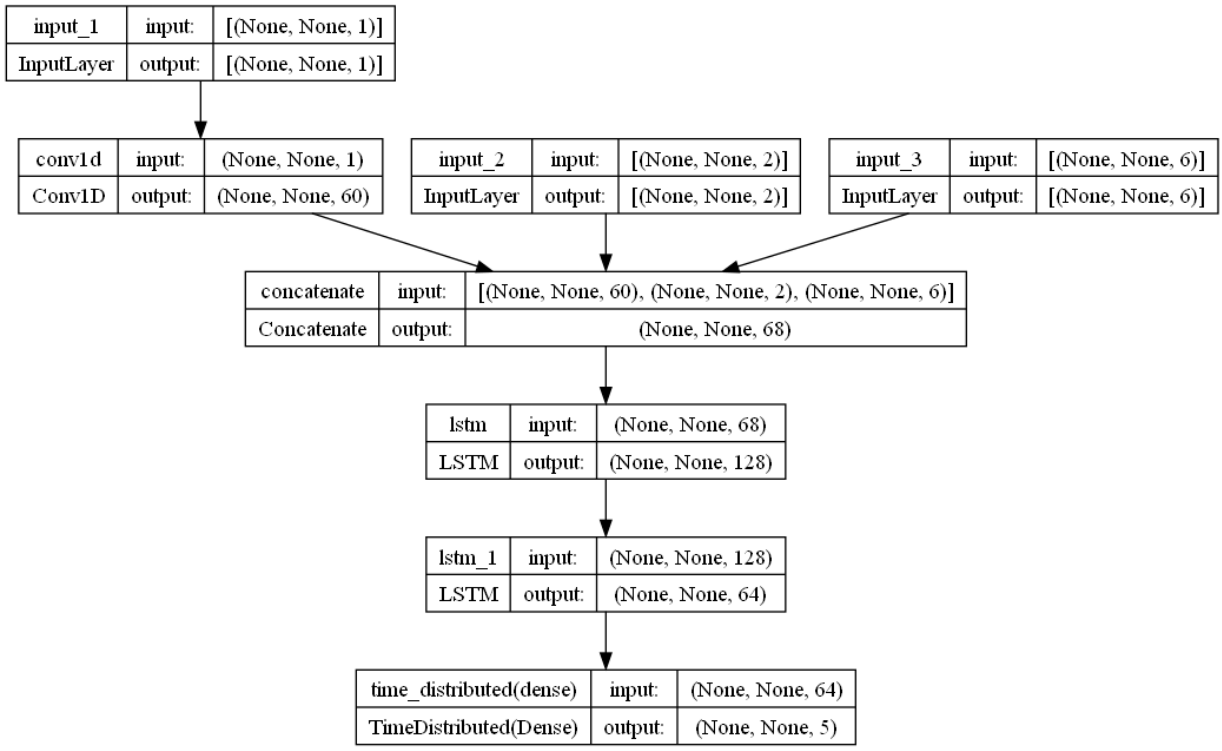


Figure 2.6: LSTM Simple Model

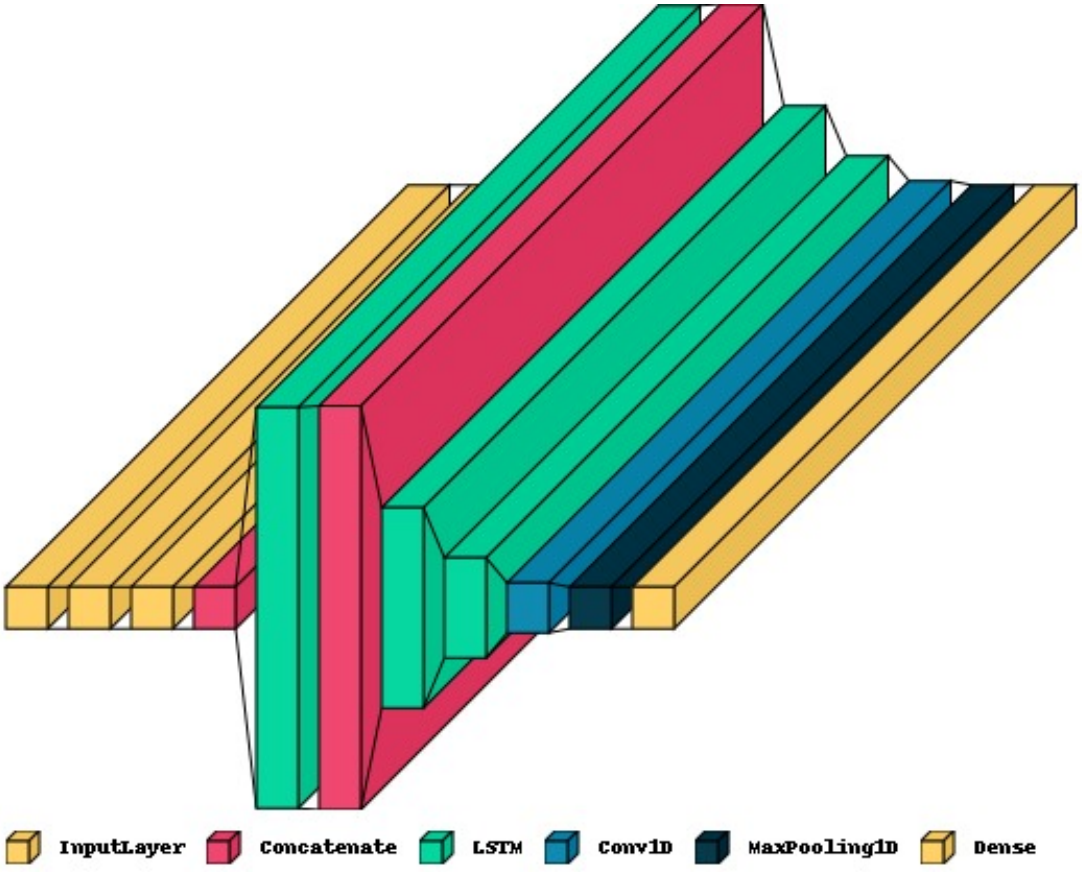


Figure 2.7: LSTM Simple Model with visualkeras

The dataset was divided into training validaton and test sets, with 70% allocated to training, 15% to validation and the remaining 15% to testing. For the training process a batch size of 8 was used in order to connect 8 different ETFs historical closing prices.
The learning rate was set to $1x10^{-4}$ for training the model. This value was chosen to ensure a stable and gradual learning process. A small learning rate allows the model to make small adjustments to the weights during each update, reducing the risk of overfitting and overshooting the optimal values. This is particularly important in time series forecasting tasks, where small changes in predictions can have a significant impact on the overall accuracy. After some trials, the conclusion is that by using a learning rate of $1x10^{-4}$, the model can converge more efficiently, potentially leading to better generalization and more accurate predictions.

2.3 LSTM and Convolutional Model

The objective of this chapter is to delve into the implementation of the hybrid model that combines Long-Short Term Memory (LSTM) network with the Convolutional Neural Network (CNN).
The reason behind this approach is to leverage the strengths of both LSTMs and CNNs to enhance the model's ability to capture complex patterns in time series data. LSTMs are particularly effective in learning temporal dependencies, making them ideal for forecasting tasks where the temporal dependencies of data play a critical role. On the other hand, CNNs detect patterns and local features within the data, which can be important to extract meaningful representation before the learning of historical sequences by the LSTM layers.
This chapter outlines the design decisions, architectural components, hyperparameters and training procedures used to develop the model. Additionally, it explores the specific configurations, such as the learning rate and input structures, that were chosen to optimize model performance and generalization.

2.3.1 Architecture and Hyperparameters of the Model

This model is the evolution of the simple model seen previously, it uses the power of the combination between LSTM and Convolutional Networks. Also in this case, the goal of the model is to predict the values for the next 5 days based on the past closing prices of ETF and stocks.
The input data are created in the form (samples, timesteps, features). The model receives as input 4 different tensors, the same as the previous model and in addiction one another that is a 4D tensor that contains the wavelets transformation of the values of all ETFs and stocks (input_8 in figure 2.8). A crucial step in preprocessing data is the preprocessing of the wavelets data by reshaping them into a 3D tensor concatenating the last two dimensions into one. The tensor is reshaped from its four-dimensional form to a three-dimensional structure, where the features are flattened. This transformation facilitates the application of convolutional layers, that are important for the extraction of meaningful patterns from the wavelets data. The first convolutional layer takes as input the reshape wavelet tensor and uses 300 filters with a kernel size of 5 and causal padding strategy. The activation function chosen is the hyperbolic tangent (tanh) in order to capture the non-linear relationships inherent in the data. The second con-

volutional layer is applied with 60 filters to further refine the feature extraction and is followed by a max-pooling layer that reduces the dimensionality of the output.
The other input tensors are passed through convolutional layers to enlarge the last dimension to have more nodes within the network with the meaningful features. The architecture then proceeds to combine all the input tensors outputs of the convolutions into a single concatenated tensor. This layer merges the time series data, macroenomic indicators, time cycles and wavelets features into a signle unified tensor that becomes the input of the subsequent LSTM layer.
The LSTM layer consists of 256 units and it returns sequences that allows the model to capture temporal dependencies in the data. The output of the LSTM layer is then concatenated with the input tensor of the time series data of ETFs and stocks in order to form a residual connection and emphasize the importance of the time series data with respect to the other tensors and become the input of two subsequent convolutional layers. The first one applies 128 filters, again using causal padding and the tanh activation function, the second convolutional layer applies 64 filters with the same padding and activation function, refines the extracted features. The output of this two layers is then passed through a Dense layer, which maps the features extracted by the network to the final output space. The output is a tensor of shape (samples, timesteps, 5) where, the last dimension represents the predictions for the future time steps, as defined by the configuration.
In conclusion, this neural network architecture is a sophisticated system that effectively integrates multiple data types, leveraging advanced techniques such as wavelet transformation, convolutional layers, and LSTM units. The careful design and selection of hyperparameters ensure that the model is well-suited to the specific prediction task, offering a robust framework for extracting and synthesizing information from diverse data sources.
The architecture of the model can be seen in figure 2.8 and the visualization of the model by visualkeras can be seen in figure 2.9.

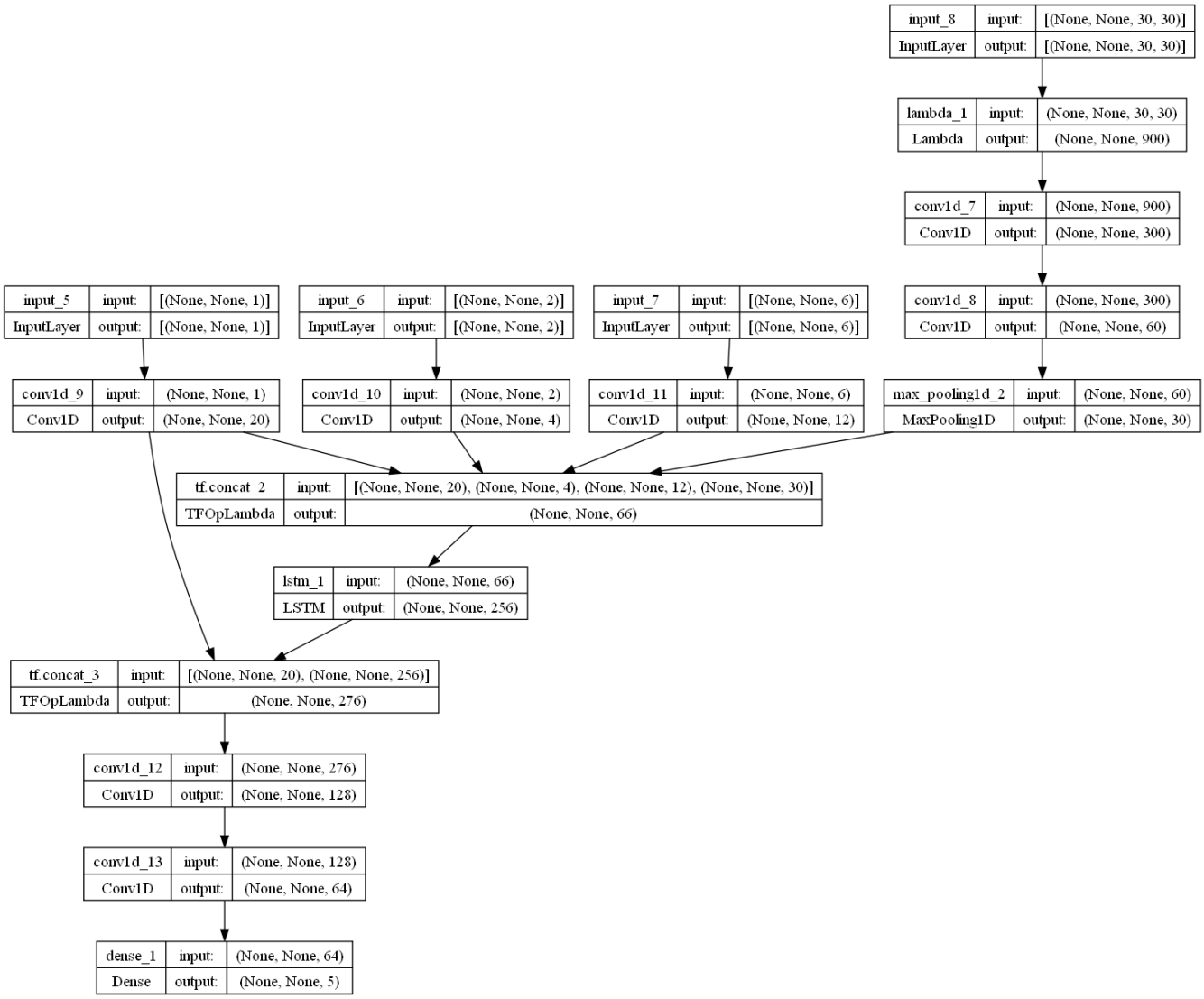


Figure 2.8: LSTM and Convolutional Model

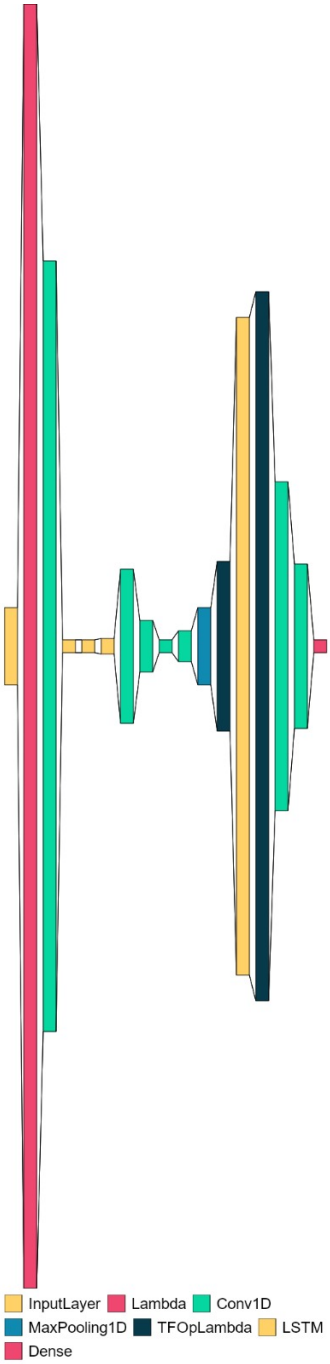


Figure 2.9: LSTM and Convolutional Model with visualkeras

For this model, the dataset was divided into training, validation and test sets, so 70% of the data are allocated for the training, 15% for the validation set and the remaining 15% for test. Initially, a batch size of 16 was selected for the training part, but after some tests this hyperparameter was set to 8, giving best results. A learning rate of $5x10^{-4}$ was set for training the model. This value was

chosen after some trials with higher and lower values, because it can ensure a gradual change of the weights of the model to optimize the learning process, to avoid overfitting and to enhance the convergence.
The final implementation of the model derive from the study of different intermediate models. It can be viewed that putting the LSTM layer first with respect to the Convolutional layers enhance the model performances. Inserting two convolutional layers for the data preprocessing of the wavelets with respect to one, can extract more robust features and the results of the model are better. Inserting also the preprocessing of all the input tensors, highlighting the last dimension incrementing the number of features is better for the model's learning. One important layer that was inserted is the residual connection, so the concatenation between the output of the lstm and the time series input data. This layer prevent the vanishing gradient problem and is significanlty important for the convergence of the model, because in deep network, very deep layers can be difficult to train because the network may have difficulty optimizing loss functions.

2.4 Loss Function

In machine learning the loss function is crucial for measuring the performance of the model. It quantifies the difference between the values predicted by the model and the true values observed from the real data. Selecting an appropriate loss function is critical in order to make the model learn the observed data.
Among all the available loss functions, the Root Mean Square Error was used for the prediction of the model, this loss function is one of the most widely used for forecasting tasks due to its sensitivity to the large errors.
The Root Mean Square Error (RMSE) is defined as the square root of the average difference between true values and predicted values squared [10].
It is defined with this formula:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

where:

- **n** are the number of observations.

- $\mathbf{y_i}$ is the actual value for the actual *i*-th observation.
- $\mathbf{y_i}$ it the actual value for the predicted *i*-th observation.

The RMSE provides a measure of the *magnitude* of prediction errors. In order to have a better fit of the model to the data the RMSE must be low.
Due to the square differences between the true values and the predicted ones it penalizes large errors more than lower ones, this important property makes RMSE useful when large errors are undesirable and need to be significantly reduced.
RMSE has also been customised to achieve better results. The idea is to give a penalty if the prediction postpones, so it goes less in the direction in with the values are going: if the signal is going up, it gives a penalty if it goes up less than it should go, vice versa, if it is going down it pushes it to go down. At this point, to not have problems with constant values that could be brought through this loss function, the results are averaged through the Loss RMSE.
This loss function produced worse results with respect to the standard RMSE, so the latter was used for all the models.

2.5 The Activation Function

Activation functions play a crucial role in neural networks, performing into hidden layers to solve complex problems and to analyze and transmit data throughout deep learning algorithms. There are dozens of activation functions, including binary, linear, and numerous non-linear variants.
The activation function defines the output of a node based on a set of specific inputs in machine learning and artificial neural networks.
For the specific topic of the thesis the **tanh** (Hyperbolic Tangent) function was used. The hyperbolic tangent function for a real number x is defined as:

$$\tanh x = \frac{\sinh x}{\cosh x}$$

where:

- **sinh**(**x**) is the hyperbolic sine of x and is defined as $sinh(x) = \frac{e^x - e^{-x}}{2}$.
- **cosh**(**x**) is the hyperbolic cosine of x and is defined as $cosh(x) = \frac{e^x + e^{-x}}{2}$.

The hyperbolic tangent function has a range of $(1, 1)$. This function was chosen because it allows for negative values, which is important given that the input data can include differences between successive days of ETF ticker values. In this way, if the difference is negative, the result can also be negative.

2.6 Callbacks

Callbacks are essentially functions that get called at certain points during the training process. Them are powerful tools that provide mechanisms to automate certain processes during the training phase of the model.
In the scripts of the model architectures there were used three important callback functions.
First one is the *TensorBoard* callback that is essential for logging metrics and visualizing the network's topology. By setting the histogram frequency as 1, the callback logs histograms of layer weights and activations at every epoch, providing insights into the model's learning process. Then, the embeddings frequency parameters logs embeddings (if any) at each epoch, which is particularly useful for visualizing high-dimentional data.
The second is the *ReduceLRonPlateau* callback that monitors, in this case, the validation loss and reduce the learning rate when the validation loss reaches a plateau. The reduce factor selected is 0.7, with the objective of enabling the model to undergo a process of fine-tuning, with the potential to achieve more favourable outcomes. The patience parameter has been set to 10, indicating that the learning rate will be reduced if there is no improvement in validation loss for 10 epochs. The cooldown parameter set to 1 adds a delay before resuming normal operations after the learning rate is reduced.
The final callback function is the EarlyStopping, which is designed to terminate the the training process if the validation loss does not improve after a specified number of epochs and in accordance with a pre-defined stop criterion. In order to prevent overfitting and conserve computational resources, this callback function was configured to monitor the validation loss. Consequently, the training process is terminated when the validation loss demonstrates a reduction in value.

2.7 Future Improvements: Graph Convolutional Network (GCN) Model

After the study of the two models seen before, LSTM and LSTM-Convolutional, there was also the study of another method that can be a possible evolution for the forecasting problem: the Graph Convolutional Network (GCN) model.
Graph Convolutional Networks represent a powerful class of neural networks designed to operate on graph structured data. This section explores the theoretical foundations and applications of GCNs highlighting the model used for the research.

2.7.1 Graph Convolutional Networks (GCNs)

Graph Convolutional Networks (GCNs) is the most prevalent and broadly applied model type of Graph Neural Networks (GNNs)[3]. Using a process called message passing, in GNN data are organised accordingly to the graph theory. Message passing embeds into each node information about its neighbors.
In GNNs, data points are called nodes, which are linked by lines called edges with elements expressed mathematically so machine learning algorithms can make useful predictions at the level of nodes, edges or entire graphs[5].
The basic building block of a Graph Convolutional Network is the Graph convolution filter. Given **S** and filter coefficients H_k the equation for the graph convolution is the following polynomial equation:

$$H(S) = \sum_{k=0}^{\infty} h_k * S^k$$

Applying this filter **H(S)** to the signal **x** obtain the result:

$$y = H(S)x = \sum_{k=0}^{\infty} h_k * S^k * x$$

[3]GNNs are neural networks that apply the predictive power of deep learning to rich data structures that depict objects and their relationships as points connected by lines in a graph.

This filter consider the summation from 0 to ∞, but in practice only a finite amount of diffusion states $K-1$ are observed, then the result become:

$$y = h_0 * S^0 * x + h_1 * S^1 * x * ... * h_{\text{K-1}} * S^{K-1} * x = \sum_{k=0}^{K-1} h_{\text{k}} * S^k * x$$

After that, the graph convolution filters aggregates the information from local to global information by using a linear combination of the elements of the diffusion sequence $S^k x$ weighted by the filter coefficients **h**.
The learning phase of the GCN is equivalent to an algorithm that learns the filter coefficients **h**. So, let $D = \{(x_{\text{i}}, y_{\text{i}}, S_{\text{i}})\}$ be the dataset of input signal x_{i}, output signal y_{i} and the graph signal operator S_{i}, during the training, the loss function L minimize the difference between the predicted output signal $\hat{y}$ and the correct output y in order to find the optimal values for the filter coefficients **h**:

$$h^* = argmin_{\text{h}} \sum_{(x,y,S)\in D} L(\hat{y}, y)$$

2.7.2 Architecture and Hyperparameters of the model

A more complex architecture has been designed for the challenging task of predicting future values of closing prices of ETFs and stocks. This model was designed in order to address the fact that ETFs and stocks are connected by their sector of interest.
Also this model uses the previous 20 days of closing prices of ETFs and stocks for the training part and predicts the next 5 days with respect to the current one. The input data are structured in five tensors to accommodate the different information sources. The first tensor (input_1 in 2.10) captures the time series data of daily closing prices of ETF tickers and stocks market, incorporating the 20-days lookback for a contextual understanding of past observations. The second tensor (input_2) represents the adjacency matrix essential for the GCN layer which encourages efficient graph-based learning. The third tensor (input_3) represents the time cycles, so the weekday and the month-day values repeated for all ETFs and stocks. The fourth tensor (input_4) captures the 6 macroeconomics indicators values of interests. The last tensor (input_5) is a 4D tensor that captures the wavelets information of all the input ETFs and stocks. This tensor represents, as said in Chapter 2, the continuous wavelet transform with 30 as the number

of logarithmic frequencies into an interval from 10 to 100. The 4th dimension of this tensor encapsulates the values of the 30 days before the current one.
Those input data are created with the form (timesteps, samples, features), where the timesteps are the historical days of interests and the samples are the ETFs and Stocks. This structure was employed to ensure that the batch size is directed toward the timesteps, enabling a more cohesive integration of different samples. By structuring the data in this manner, not only the temporal connections within the time series data are facilitated but also allowed the Graph Convolutional Network (GCN) to effectively link different samples.
The architectural complexity involves a sequence of operations. An initial convolutional layer for the time cycle tensor and for the macroeconomic tensor is used as a robust feature extractor in order to increase the size of the features, employing a kernel size of 5 and a causal padding to preserve temporal order.
A reshape layer, also called Lambda layer, is used for decreasing the wavelets tensor from 4D to 3D encapsulating the last two dimensions into one.
Then, two GCN layers leverage the adjacency matrix to facilitate graph-based learning which allows the model to identify complex relationships within the data. The first GCN layer takes as input the adjacency matrix and the wavelets tensor, the second takes as input the adjacency matrix and the time series values tensor. The results of these operations are then concatenated into a single tensor to continue the fitting.
Convolutional layers follow, designed to decrease gradually the last dimension of the tensor, representing the period of 5 days of forecasting. After these convolutional layers there is a max pooling layer that is also used for effectively reducing the spatial dimensions of the feature maps, thereby lowering computational complexity and the number of parameters in the network. This reduction helps prevent overfitting by focusing on the most significant features while disregarding irrelevant details. Additionally, the max pooling layer introduces translation invariance, enabling the model to recognize patterns regardless of their position in the input data. Overall, this layer enhances the efficiency and robustness of the convolutional neural network. The model utilizes a dense layer for the final transformation of learned representations into output predictions.
The dataset was divided into training, validation and test sets, with 70% allocated to training, 15% to validation and the remaining 15% to testing. For the training process, a batch size of 16 was selected after experimentation with larger batch sizes, which did not yield satisfactory results in terms of model

performance.
In the initial stages of training, a relatively high learning rate was employed to speed up convergence. However, as the training progressed, the learning rate was systematically reduced based on observations in TensorBoard. By monitoring the loss curves and other relevant metrics, it became evident that lowering the learning rate gave an important contribution on stabilizing the model's performance and improving accuracy. This gradual reduction helped in fine-tuning the model, allowing it to settle into a more optimal set of parameters as it approached convergence. Additionally, this approach helped prevent the model from overshooting the optimal parameters, ensuring a smoother and more controlled descent during the latter stages of training. The architecture of the model can be seen in the figure 2.10 and the visualization of the model by visualkeras can be seen in figure 2.11.

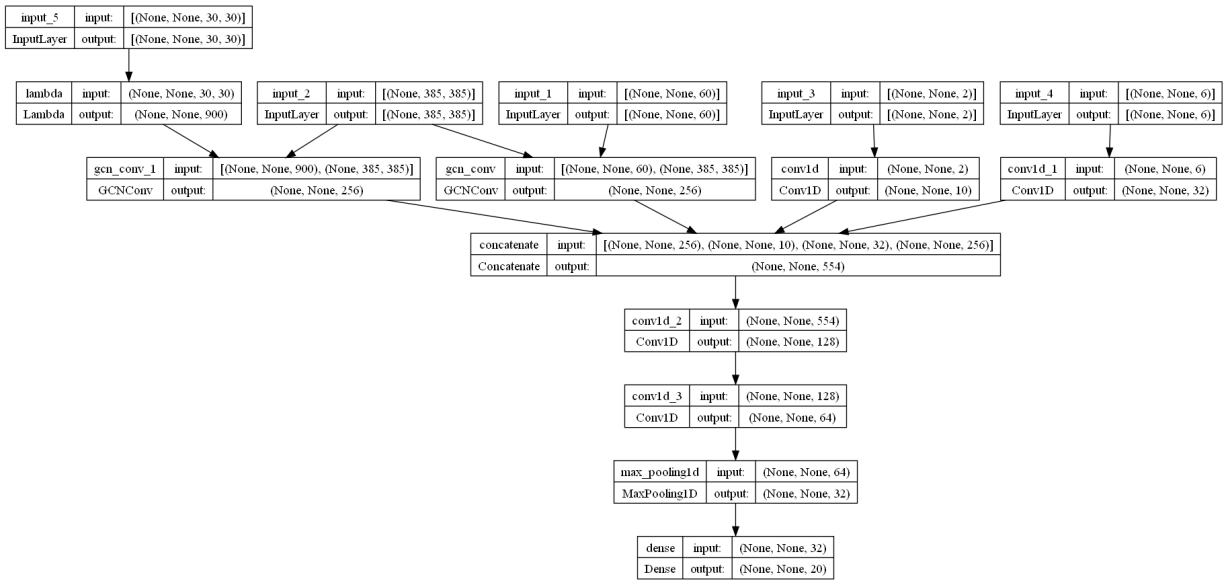


Figure 2.10: GCN Model

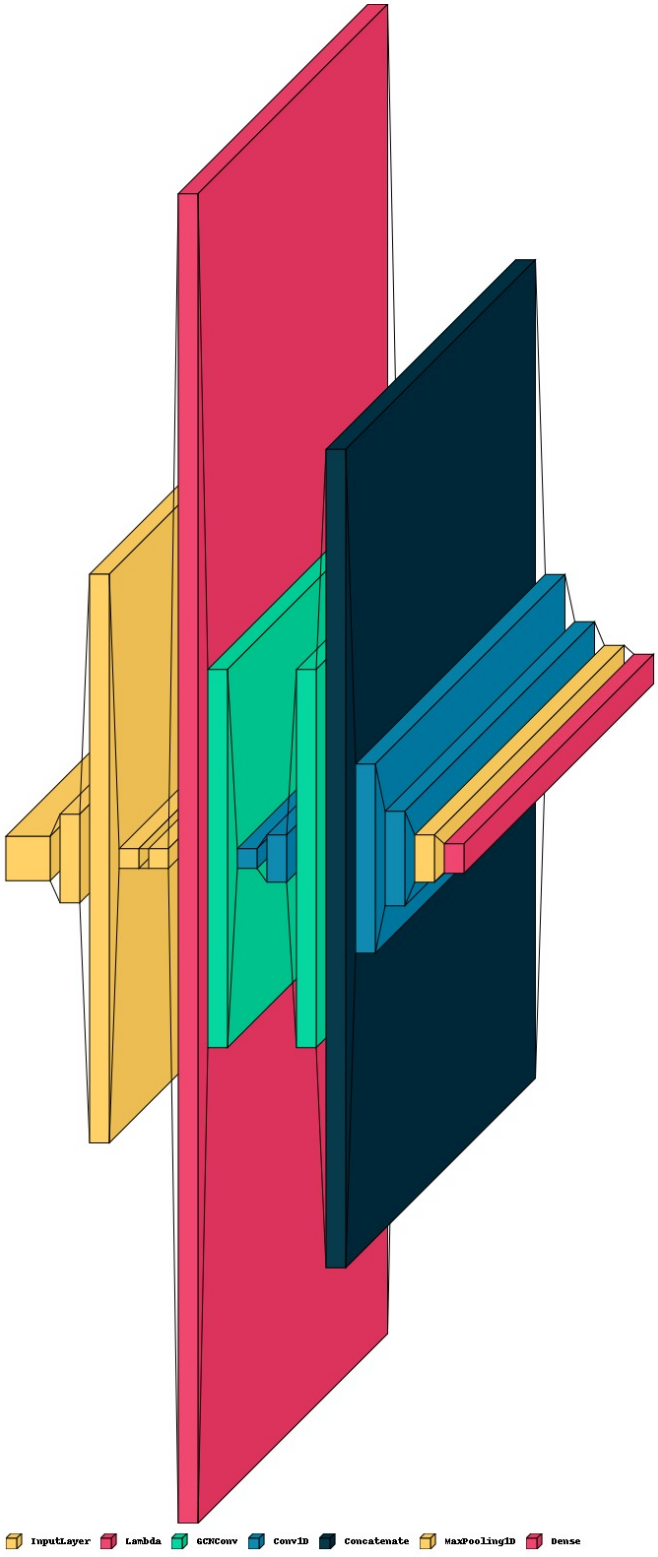


Figure 2.11: GCN Model with visualkeras

Some tests were done to try to enhance this model's results introducing several modifications. Initially, convolutional layers were added to the input data prior to passing them through the Graph Convolutional layers in order to extract relevant patterns[9]. For the wavelet-transformed data, switching from Lambda layer to Reshape layer enhance the model's performance. Then, an LSTM layer was introduced after the Graph Convolutional layer to the input data relating to the closing price of the ETFs and Stock markets. The purpose of this was to capture the temporal dependencies derivating from the nature of the stock prices. A residual connection was introduced to the final dense layer taking the information of this LSTM layer in order to emphasize the importance of the closing prices with respect to the other input. By combining this modifications the model obtained better results, but they did not come close to the expected ones. The RMSE of the predictions was reduced compare to the initial model, but was higher than desired. These changes represent a valuable first step in enhancing the model and provide a foundation for future works. In figure 2.12 there is the architecture of this final model.

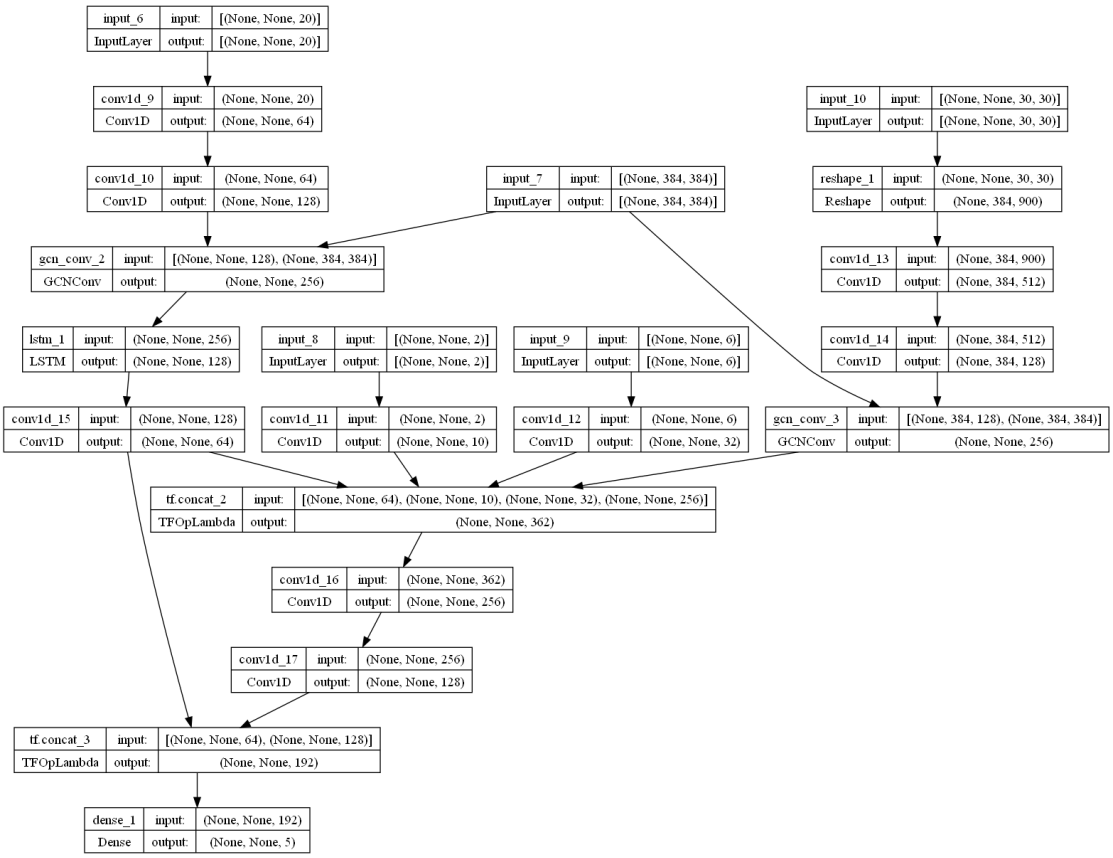


Figure 2.12: GCN final Model

2.8 Future Improvements: Graph Neural and LSTM network

The GCN model, while benefiting with the integration of convolutional layers and the other modification seen above can have significant improvements. The model's performance has not reached the desired results so, this section explores potential directions for future works that can enhance the accuracy leveraging to better results.
The objective is to present an exploration of the Keras-based time series traffic forecasting model and its adaptation to the problem of this thesis[4].

2.8.1 Overview of the original model

The original Keras example for time series traffic forecasting uses both LSTM and GCN in order to extract spatial dependencies and temporal correlation. This model is used to predict traffic flow in urban networks, but can be enhanced for forecasting ETF closing prices by adapting the latter data in the way in which the former are preprocessed. The model's architecture begin with the input preprocessing in which data are normalized and split into a sliding window of input features and corresponding target outputs. The objective of this is to permit the model to predict the next time step based on the previous window of observations. This method, ignores the dependency of the traffic speed of one road segment on the neighboring segments. After that, the data are divided, as usual, in train, validation and test sets for the learning. The adjacency matrix is computed by using the road segments distances, the assumption is that there is an edge between two nodes in the graph if the distance between the corresponding roads is less than a threshold. The inputs are then passed through the Graph Convolutional layer that performs the following steps: it computes the nodes' representation by multiplying the input features by a weight to give the correct dimensions, it aggregates neighbors' message by first aggregating the neighbors' representations, then it multiplies the results by the weight and it computes the output by combining the nodes representations and the neighbors' aggregated messages. In this example the input to the layer is a 4D tensor of shape (samples, timestemps, input_seq_length, in_feat) where input_seq_length is the number of last values of the speed in each road that is considered and in_feat is the number of features to be considered. By applying the graph convolution layer

the output is another tensor containing the nodes' representations over time computed by using also the information of its neighbors. The output is then passed through an LSTM layer to process the information over time capturing long-term temporal dependencies in the data. Finally, a dense layer produces the prediction for the next time step in the series.
In the example, the performances of this model are compared to a naive forecast that takes the last value of the speed for each node. The performance are evaluated by using the mean absolute error (MAE) that is a measure of errors between paired observations expressing the same phenomenon and is calculated as the sum of absolute errors (the Manhattan distance[4]) divided by the sample size and the formula is the following: $\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - x_i| = \frac{1}{n}\sum_{i=1}^{n}|e_i|$. The model has obtained optimal results almost reaching the naive model. To improve the model's accuracy all model hyperparameters should be tuned carefully or also several LSTM and GC blocks can be inserted in stack to increase the representation power of the model.

2.8.2 Adapting the model to Financial Time Series Forecasting

Adapting the model to financial time series forecasting requires several key modifications because the data are different and there is a risk of not obtaining results suitable for the research. This types of data differs from the time series traffic data in terms of volatility and the complex interactions between market variables. One important adaptations to make is to add to the input also the other economic variables that are used also for the other models: the time cycles, the macroeconomic indicators and the wavelets if needed. In financial forecasting it's critical to incorporate these variables and so first the data engineering requires these modifications. The idea is to preprocess them using normalization techniques such as MinMax scaling, ensuring that all features had a comparable range and did not exploit during the learning process.
The CNN layer in the original model was created for capturing the spatial patterns in the traffic data, in this context the convolutional layer can be used to detect patterns like short-term price movements. Also in the other models the convolutional layer help to do that enhancing the performances. This convo-

[4]The Manhattan distance between two points is defined to be the sum of the absolute differences of their respective Cartesian coordinates

lution is into the graph convolutional layer, this one can be adapted inserting a new adjacency matrix that takes into account the sector to which the tickers and stocks belong to indicate their proximity. Given the importance of these patterns in financial markets, this layer's kernel and size must be fine-tuned to detect the short-term fluctuations in an efficient way. The LSTM layer is crucial for modeling temporal dependencies. This is amplified in financial time series forecasting where long-term dependencies and historical market changes have an impact on future trends. Some adaptations can be done here: increase the number of units allowing the model to look back to a larger historical context and add multiple LSTM layers in stack to capture the relations between the time series data.
It can be used also a similar strategy for residual connections in order to preserve the closing prices during the forward pass of the network. In this way, it ensures that the closing prices are always accessible to the final dense layer. Another way that can be used is to add one or two convolutional layer before the final dense layer in order to decrement gradually the dimension of the tensors during the forward pass of the network maintaining the right pattern from input to output. To evaluate the effectiveness of the adapted model, also to compare the results between different types of input the MAE can be used as performance indicator but, in order to compare the results of this model with respect to the others used for this thesis, the RMSE is the one indicator that can give the correct vision for the problem.

3

Results

This chapter represents the results obtained from the analyses conducted in the course of this research project. The objective is to present a comprehensive and detailed overview of the findings, emphasising the key aspects of interests and establishing a clear connection to the research objectives previously outlined.
To make everything clearer this chapter is divided into two sections, each representing the results of a distinct neural network model employed for forecasting purposes.
The primary objective is to evaluate the performance of these models and to compare their predictions against a baseline model, the specifics of which will be explained in greater detail later in this chapter. The results are presented in a manner that allows for the effective evaluation of each model's performance, including an assessment of its accuracy and consistency in different market conditions.
The findings offer insights into which approaches deliver the most reliable forecasts, providing a foundation for further discussion and interpretation.
In this section are highlighted, for all the models, the loss in the training set, the loss in the validation set, the loss of the test set compared with the loss of the base model of the test set. Knowing that, all the input data were transformed by using different transformers, the output predictions were transformed back into real data and also the loss of them was calculated. Here, there can be found also different figures representing the predictions of the real data compared to the real input data and some images of the TensorBoard showing how the models

had learned from the data.

3.0.1 Base-Model

The Base-Model is a fundamental approach for forecasting short-term time series data, as said previously. Its function is to have a comparison model, in fact, the loss function that is calculated on this Base-Model is the point that this research wants to reach and then, in future, to overcome.
In the Base-model the current value is extended in a certain number of timesteps, this number is the lookforward period, as if it were unchanged. This will then be compared with the true value of the following days which is known, being a storical series.
The model receives as input a three-dimensional tensor structured as (samples, timesteps, features) or (timesteps, samples, features), where the features represent predicted values for the upcoming days, and sistematically shifts these values forward in time and computes their mean. Mathematically, for each timestep t, the predicted value is given by:

$$\hat{y}_{\mathrm{t}} = \frac{1}{lookforward} \sum_{i=1}^{lookforward} \mathrm{shift}(y_t, i)$$

where the $shift(y_t, i)$ represents the predicted value shifted by i timesteps in the future.
In figure 3.1 there is a representation of the base model applied on the test set of the ticker XLB[1] with a lookforward of 5 days. In blue there are the actual values for the ticker XLB, while in orange there is the prediction made by using the Base-Model.

[1]The ticker symbol XLB correspond to the Materials Select Sector SPDR fund. It is an ETF that tracks the performance of companies in the materials sector of the S&P 500.

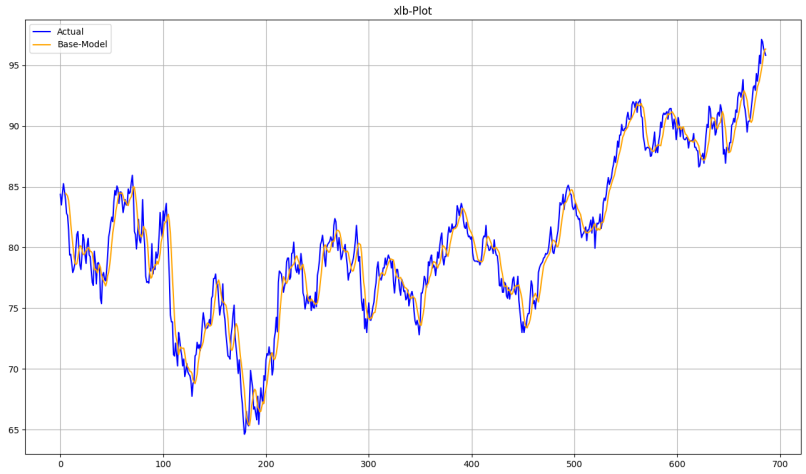


Figure 3.1: Plot of the Base Model on XLB ticker with 5 days lookforward

In figure 3.2 there is the representation of the base model applied on the test set of the stock ACN[2] with a lookforward of 5 days. Also in this case, in blue there is the representation of the actual values, while in orange there are the Base-Model predictions.

Figure 3.2: Plot of the Base Model on ACN stock with 5 days lookforward

[2]ACN (Accenture plc) is a global professional company, providing a broad range of services in strategy, consulting, digital, technology and operations.

This averaging method is useful in scenarios that can be influenced by short-term anomalies because it provided a smoothed estimate. By averaging over multiple future timesteps, the model reduces the impact of the outliers that can may be present by the using of neural network models yielding into a more reliable forecast.

3.1 LONG-SHORT TERM MEMORY (LSTM) SIMPLE MODEL RESULTS

The LSTM simple model was the first approach tested and it is very important for discovering the feasibility of the prediction of financial time series. This model was developed with the aim of capturing patterns in historical data and evaluating its capability in forecasting short-term trends. The input data are divided in three primary tensors: historical closing prices of the ETFs and stocks, time cycles transformations of time series data and a range of macroeconomic indicators. The model was trained with the objective of predicting price trends for the subsequent five days based on past performance.
The primary focus when this model was developed was on understanding its behaviour and identifying any limitations or challenges in short-term forecasting. The results had a chance to prove valuable insights into how this architecture performs even with the complexity of financial data and the limited input window. Another important aspect was the understanding of the power of LSTM to learn historical series given the nature of the latter.
This initial implementation play a crucial role in setting the direction for subsequent research and model implementations even if it simplicity. The results obtained helped identify key challenges, such as identifying crucial patterns and capturing information on how to enhance future models.
Overall, this LSTM model serves as a foundation for more advanced exploration, offering a first vision of the potential and limitations of using deep learning in financial forecasting. The insights gained from this preliminary experiment set the stage for future improvements in both model architecture and feature engineering, driving the research on the development of more sophisticated predictive systems capable of dealing with the complexities of financial markets.

3.1.1 Result Analysis

The results obtained from the LSTM simple model, as expected, show limitations in accurately predicting future trends over the short period of five days. While the model is able to capture some general trends in price movements, the predictions often show significant shifting from the actual values.
During the training of the simple LSTM model, TensorBoard proved essential for real-time monitoring of the training and validation loss curves. The observed curves show a consistent and gradual downward trend without any noticeable "jumps" or fluctuations, indicating that the model is learning effectively. The decrease in both training and validation loss suggests that the model is successfully learning patterns in the data and generalize them well, without significant overfitting and underfitting. This behaviour confirms that the model improve its predictions as training progresses. The figure 3.3 shows the training and validation loss curves in TensorBoard in which can be seen the highlighted considerations. In green there is the validation loss curve, in red there is the training loss curve.

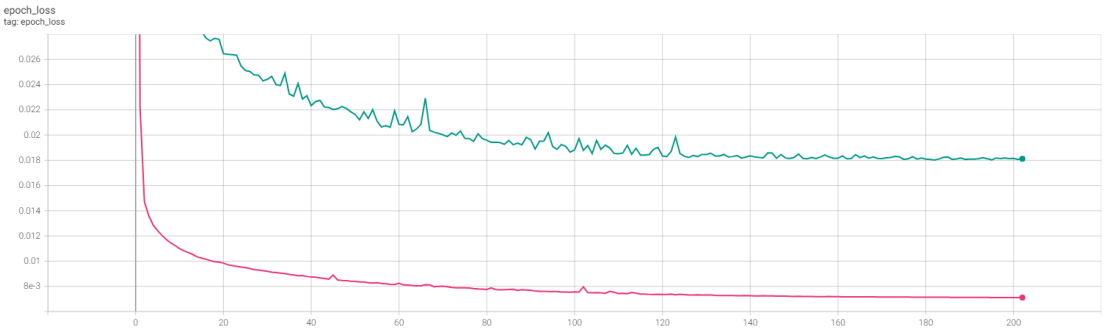


Figure 3.3: Curves of Training and Validation Loss functions in TensorBoard

In general, the model tends to follow the overall direction (upward or downward) of the market but tens to struggle with precision in terms of timing of price fluctuations. The behavior is likely due to the limited input window and the inherent noise in financial time series data. Additionally, the model's performance varies significantly depending on the type of transformation that is carried out on the different inputs, to make them more coherent for the learning of the model.
The results show that the predicted data appears to be shifted compared to the actual price movements. While the model seems to recognize general trends, these are often delayed or advanced, leading to misalignment between the pre-

dicted and real price points. This misalignment is evident in the visualizations, where it's clear that even if the model recognizes the overall direction, the timing is off resulting in not optimal performance.
In terms of input transformations, it was observed that the MinMax scaling is the most effective approach. Techniques like power transforms were tested, which should ideally benefit the model because they normalize the inputs and enhancing learning, but they didn't yield better results. Even when using techniques to shift the inputs (effectively computing the derivatives), the model does not produce accurate predictions. This suggests that even their theoretical advantages, these transformations introduce complexities that this simple model can't completely address. Below are reported some images that show the results with real data of some ETFs and stocks which are considered important to to have a complete vision of the correctness of the model, but above all the limitations.

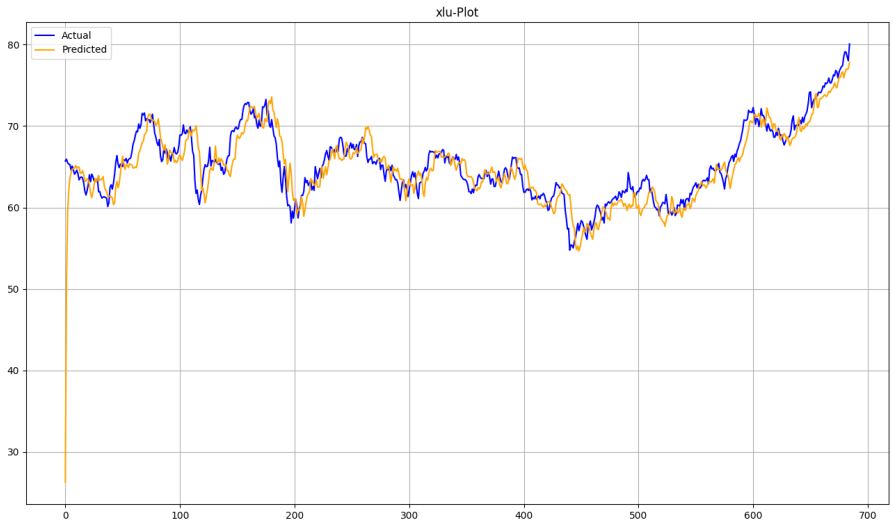


Figure 3.4: Results of the model for the ticker XLU

In figure 3.4 there is the output of the model with the ticker XLU, in blue there are the real historical data and in orange there are the predicted ones. The figure shows that in the first part the predicted data start from zero and then they settle to the real data, this is the consequence of the using of LSTM. In fact, the LSTM start from a value of zero and during the training settles the predictions to the data that it receives in input. This results are obtained with the MinMax transformation of the input. As said previously, the predicted data

are shifted with respect to the real ones, the model learns and recognizes the rise and the fall of the tickers' prices, but it seems that these changes are not consistent with the timing of the real data, but they are still delayed.
In order to make a comparision between the predictions of ETFs and stocks, in figure 3.5 there is the output of the model for the stock of AMZN, also in this case in blue there are the real historical data and in orange the predicted data. AMZN stock is the stock of the company Amazon that is one of the largest and most influential companies globally, it dominates sectors like e-commerce, cloud computing, digital advertising and more. In the figure are shown the results obtained with the MinMax transformation without the shifted version of the values in input. Also in this case it can be seen that the model follows the general trend with a delay in the time of predictions. This is the most important limitation of the model.

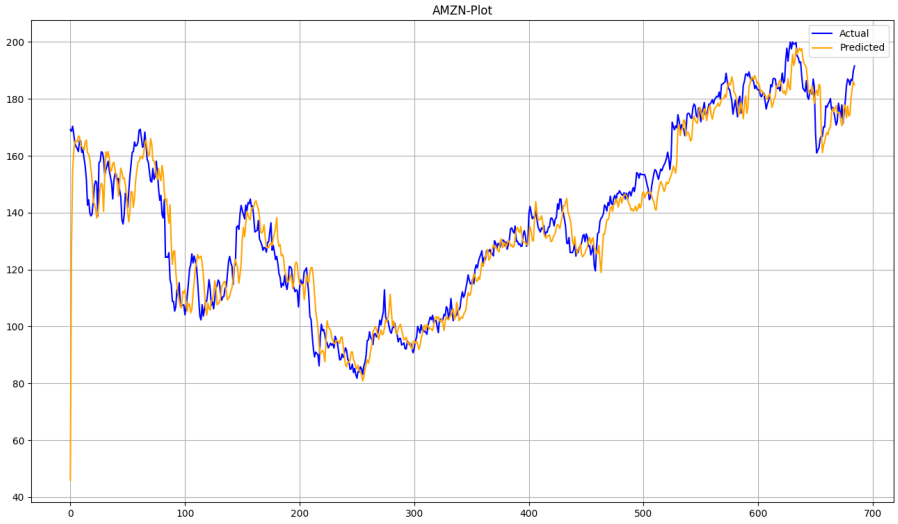


Figure 3.5: Results of the model for the stock AMZN

To emphasize the presence of a major limit, such as the shifting of the forecast data compared to the current ones, two images are printed below which show an enlargement of the results obtained by the model with respect to a specific time interval. The figures represent the outputs regarding the XLU ticker (figure 3.6) and the AMZN stock (figure 3.7).

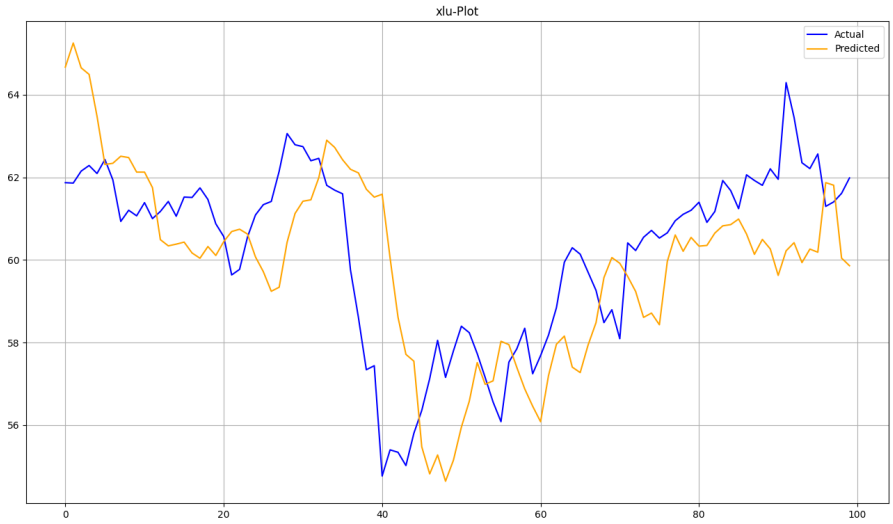


Figure 3.6: Enlargement of the results of the model for the ticker XLU

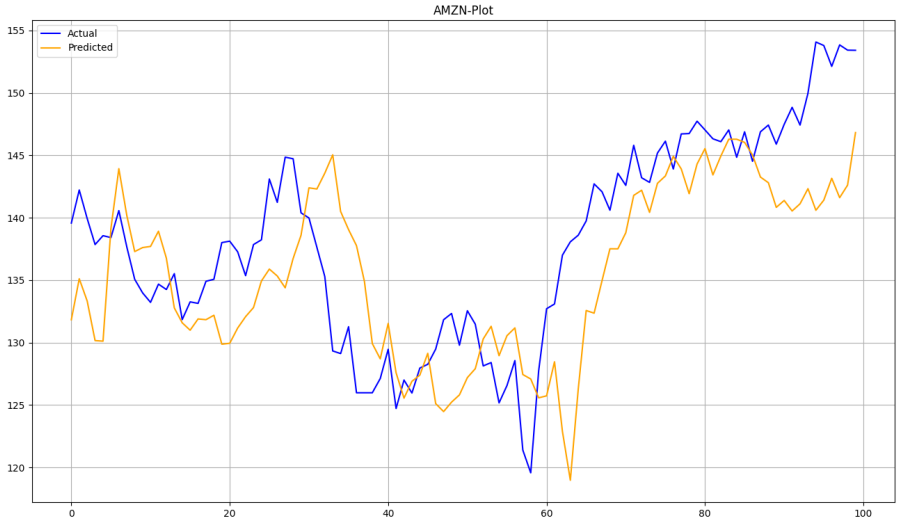


Figure 3.7: Enlargement of the results of the model for the stock AMZN

In conclusion, the model is behaving like the Base-Model guessing that the most likely value is the last one.

3.1.2 Performance Evaluation

This section analyzes the performance of the model by using different pre-processing methods examining the Root Mean Square Error (RMSE) in the stages of the learning process: training, validation and test. In order to do a complete examination this section also analyses the results comparing them with the baseline model.

Input Transformer	Shift	RMSE training	RMSE validation	RMSE test	RMSE Base-Model Training	RMSE Base-Model Validation	RMSE Base-Model Test
MinMax	No	0.0071	0.0174	0.0225	0.0052	0.0124	0.0147
MinMax	Yes	0.0495	0.1181	0.1450	0.0215	0.0530	0.0623
PowerTransformer with Yeo-Johnson	No	4.6493	2.4335	5.7892	0.2517	0.2089	0.3716
PowerTransformer with Yeo-Johnson	Yes	0.7475	2.2819	2.9720	0.6874	2.1665	2.6745

Table 3.1: LSTM Simple model Results

In table 3.1 are reported the mentioned results. As it can be seen, the Power-Transformer with Yeo-Johnson did not provide acceptable results, the RMSE is too high to consider them into an useful research. MinMax scaling without the shift operation seems to provide the best performance with the lowest RMSE accross all stages. Even if the theory predicts that the use of the Power Tranform, introducing a Gaussian value to the data and making them more normal-like, this model did not benefit from it, on the contrary, using the MinMax scaling, therefore transforming the data following a certain range of values, brings more benefit. The objective of equaling or exeeding the base model was not achieved, but it can be considered as an initial model from which to start for improvements. The results with MinMax scaling without the shift operation are quite satisfactory but can't be used for a real world problem.

3.2 LSTM-CONVOLUTIONAL MODEL RESULTS

In this section are described the results of the LSTM-Convolutional model that was developed trying to increase the results searching from the errors of the first model. The LSTM layer is used to forecast the short-term trends, then the convolution extracts the fundamental features that will be then used in the dense layer that gives the final results. The input data consisted of four important tensors: historical closing prices of the ETFs and stocks, the time cycles transformation, a range of the macroeconomic indicators and the wavelets transformed data. Wavelets transformed data are crucial in order to recognize the periodicity that may change in time. The model was trained with the objective of predicting price trends for the subsequent five days based on the past performance.
The model was developed with the aim of understanding whether the power of LSTMs could be enhanced with the use of convolutional layers, trying to extract the best possible features from the data, increasing the inputs and therefore creating a more complex model.

3.2.1 Result Analysis

As for the previous model, also this model's results show limitations in accurately predicting future trends. It captures the trends in price movements, but the predictions are shifted with respect to time from the real values, so there are some differences.
The training and validation loss curves observed by the using of TensorBoard show a gradual downward trend, there are some "jumps", but in line with expectations, so the model learn effectively, it uses the first 50 epochs to set the value and then continue the training. In the end, the model learn effectively and generalize well, without any significant sign of overfitting or underfitting. In the figure 3.8 in grey can be seeing the training loss curve and in orange the validation loss curve highlighting the previous observations.

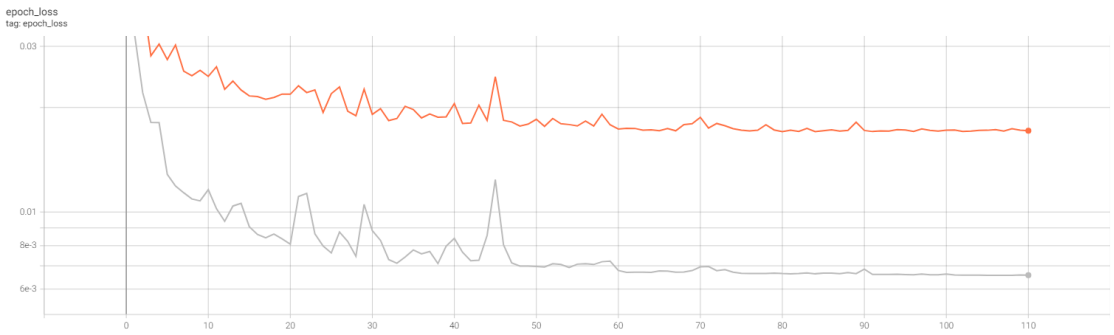


Figure 3.8: Curves of Training and Validation Loss functions in TensorBoard

The results of this model are in line with those of the previous model, them indicate that there are still the same problem as before. The model follows the market trends with precision but also in this case the values vary in terms of the timing of price fluctuations. Depending on the type of transformations applied to the input data, the model's performance varies as with the previous model. The predicted data are shifted compared to the real ones, in line with what was said regarding the model without convolutions. The using of wavelets transform on the historical prices of the ETF enhance the model's accuracy. Wavelet transforms are particularly effective in capturing both time and frequency domain characteristics, allowing the model to better detect and adapt to underlying patterns and trends in the data. The MinMax transformation of the input data once again is the most effective in terms of results, with the PowerTransformer with Yeo-Johnson that still presenting itself as a transformation that does not produce acceptable results. The derivative techniques to shift the data does not produce accurate predictions, so maintaining the original data and transforming them into a particular range with the MinMax transformation is the best practice for this model and for those types of data.
The following two images show the results of the LSTM-Convolutional model for the forecasting problem that support the reasoning discussed in the previous lines. The predicted data are shown in orange, while the real data are in blue. Figure 3.9 represents the results obtained with the MinMax transformation of the input values with the ticker XLB, while in figure 3.10 there are the results of the model with MinMax transformation of the input values with the stock ALGN. Align Technology, listed on NASDAQ as ALGN, is a leading medical device company primarily known for its Invisalign system of clear aligners.

Figure 3.9: Results of the model for the ticker XLB with the MinMax input transformation

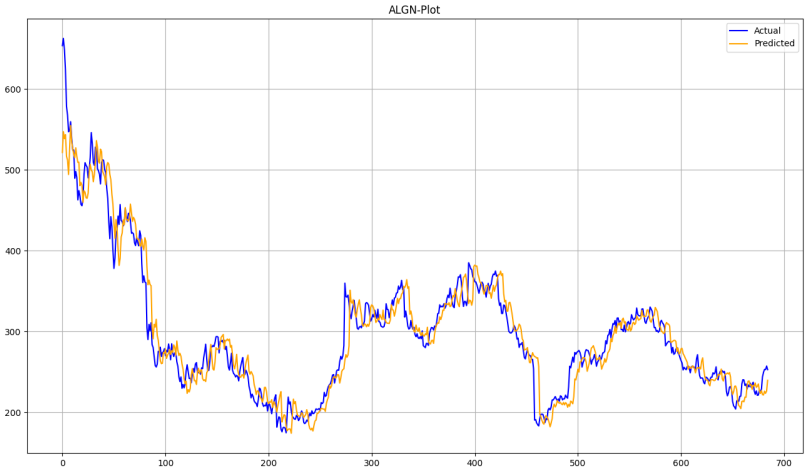


Figure 3.10: Results of the model for the stock ALGN with the MinMax input transformation

Also in this case, to emphasize the presence of the shifting of the forecast data compared to the current ones, the two images below show an enlargement of the results obtained by the model with respect to a specifigc time interval. The figures represents the results obtained by the same ticker an stock: XLB in figure 3.11 and ALGN in figure 3.12. This results can be considered good, it can

be seen that shifting, although present, has decreased compared to the previous model.

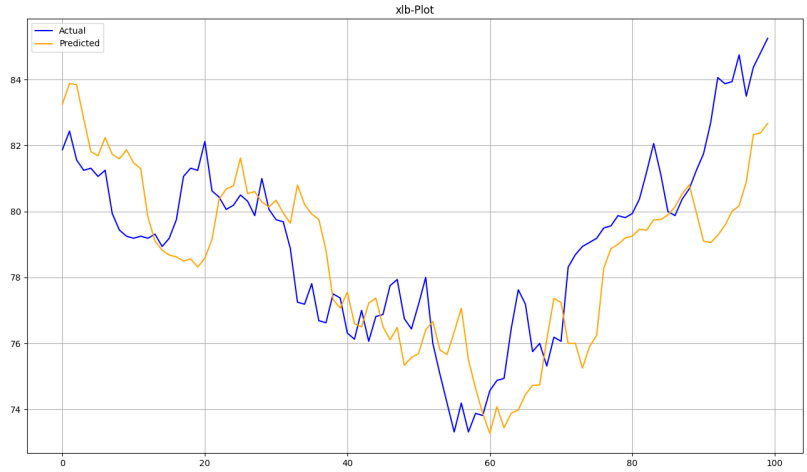


Figure 3.11: Enlargement of the results of the model for the ticker XLB

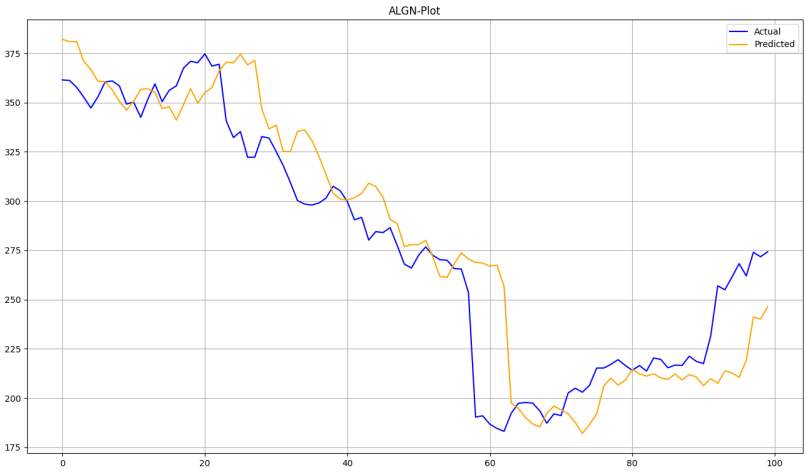


Figure 3.12: Enlargement of the results of the model for the stock ALGN

3.2.2 Performance Evaluation

In this section are analyzed the performance of the LSTM-Convolutional model. Below is reported the table that contains all the RMSE of the model accross different stages of the execution: training, validation and testing. Also

for this model, different transformations of the input data were tested in order to determine which one is the best to use.

Input Transformer	Shift	RMSE training	RMSE validation	RMSE test	RMSE Base-Model Training	RMSE Base-Model Validation	RMSE Base-Model Test
MinMax	No	0.0066	0.0170	0.0205	0.0052	0.0124	0.0147
MinMax	Yes	0.0487	0.1196	0.1467	0.0215	0.0530	0.0623
PowerTransformer with Yeo-Johnson	No	4.8817	2.6820	6.1023	0.2517	0.2089	0.3716
PowerTransformer with Yeo-Johnson	Yes	0.7134	2.3485	2.977	0.6874	2.1665	2.6745

Table 3.2: LSTM-Convolutional model Results

Table 3.2 shows that the MinMax transformation without the shifting of the input data yield to the best results. This model does not produce accurate results for the PowerTransformer with Yeo-Johnson, which again shows that this transformation cannot be used for this type of data. With regard to the other model, this one had lower RMSE and from the plotting of the results it appears that this is better.
The results obtained with the MinMax transformation without the shifting of the data are similar with respect to the Base-Model results, indicating that this model already performs reasonably well in this scenario. This suggests that the LSTM-Convolutional model has potential for further improvement.

4

Conclusions and Future Works

The world of financial forecasting, particularly in the context of ETFs and stock markets presents an extraordinary challenge. The objective of this thesis was to explore how neural networks can be employed to predict the closing price of ETF tickers and stock markets. The objective was ambitious: to try to achieve the performance obtained by a baseline model.

The inherent difficulty seen within ETFs intricate nature, characterized by frequent fluctuations and high volatility, created an important research on the different methods of transformation of the input data and on the various neural network models to be applied. Although the results did not fully reach the initial benchmarks, they were nonetheless encouraging and can be considered an initial approach for the real-worls applications. The forecasting of stock prices is notoriously difficult and predicting them perfectly is rarely, if ever, attainable. Also the market, with all its facets, and with all the changes it can undergo, presents an important challenge regarding its prediction, which in fact, can sometimes be almost impossible. However this considerations, the models developed throughout this research have shown that neural networks can provide meaningful insights, even if they require more study to reach a greater precision.

The most important contributions that these thesis can give are that neural networks, particularly LSTM models and LSTM-Convolutional hybrid models can capture the trends in financial data, but they are not precise in capture the short-term fluctuations. This research showed that the LSTM learned the historical price patterns but it faced limitations in timing accuracy, predicting

shifted output. This results is important because it open some considerations on how temporal dependencies and the complexity of financial time series can be processed in order to train better the models.
Additionally, the thesis highlighted the crucial importance of the data preprocessing. In this way, techniques like MinMax scaling provided the best results for normalizing input data, while more complex transformation, like power transformers, introduced some noise without giving better results. Also this finding is important, because it shows how well-chosen preprocessing techniques may yield to better results.
From the study it can be seen that using a convolutional layer as input layer enhances the performance of the model. Additionally, the use of wavelets has improved the precision of the model, potentially due to their ability to capture both frequency and time-domain information. It is also evident that applying a reshape layer in the GCN architecture provides better performance compared to using a lambda layer. Furthermore, the use of max pooling layer helps to reduce the data dimensionality while maintaining important features, contributing to a more robust model.
There are several directions for future research that could enhance the performance of the models developed for this thesis. First, as said in chapters 2.7 and 2.8 the Graph Convolutional Networks can be potentially improved by enhancing the model's ability to understand the sector-based dependencies and the market relationships in order to capture temporal and spatial patterns. The hybrid approach of GCN and LSTM can potentially improve the model's ability to forecast short-term trends. One another important future works can be increasing the size of the dataset and exploring different types of economic indicators in order to provide more context for the models, helping them to better anticipate market shifts. Finally, the fine-tuning of the hyperparameters such as the learning rate, batch size, and the number of units for the layers of the model could lead to better performances.
In conclusion, the results of this thesis show that neural networks can be useful for predicting market trends, even if the financial forecasting problem remains complex and with uncertainty. The models developed in this work provide a foundation for further exploration and refinement, them can give useful insights into how deep learning can be applied to the financial sector.

References

[1] Wikipedia contributors. *Stock market*. 2024. URL: https://en.wikipedia.org/wiki/Stock_market.

[2] Sergio Lopez Dubon et al. *A Machine Learning Approach for Tidal Flow Classification*. School of Engineering, The University of Edinburgh. Edinburgh, EH9 3DW, UK. 2023.

[3] IG Group. *What Are the Key Macroeconomic Indicators to Watch?* 2019. URL: https://www.ig.com/en/trading-strategies/what-are-the-key-macroeconomic-indicators-to-watch--191014.

[4] Keras. *Time Series Forecasting with CNNs and LSTMs in Python using Keras*. https://keras.io/examples/timeseries/timeseries_traffic_forecasting/. 2024.

[5] Rick Merritt. *What Are Graph Neural Networks?* URL: https://blogs.nvidia.com/blog/what-are-graph-neural-networks/.

[6] S. N. Sivanandam S. Kumar Chandar M. Sumathi. *Prediction of Stock Market Price using Hybrid of Wavelet Transform and Artiicial Neural Network*. URL: https://www.researchgate.net/publication/298330331.

[7] scikit-learn. *MinMaxScaler*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html.

[8] Inc. The MathWorks. *What is a Wavelet?* 2024. URL: https://it.mathworks.com/help/wavelet/gs/what-is-a-wavelet.html.

[9] Ege Alp Turkyener. "Development and Calibration of a Graph Neural Network for Sewage Water System". Master's Thesis. Università degli Studi di Padova, 2023.

[10] *What is Root Mean Square Error?* URL: https://c3.ai/glossary/data-science/root-mean-square-error-rmse/.

[11] Russell Wild. *Bond Investing for Dummies*. 3rd. Wiley, 2020.

Acknowledgments

First of all, I would like to express my deepest gratitude to my parents for always supporting me throughout this journey. Without them, none of this would have been possible.

I would also like to thank my friends for always believing in me. Growing up together, I have learned the true meaning of Friendship. Thank you.

Lastly, I would like to thank my entire family, who has always been a source of motivation.