UNIVERSIDAD DE INVESTIGACIÓN DE TECNOLOGÍA EXPERIMENTAL YACHAY

Escuela de Ciencias Matemáticas y Computacionales

Forecasting time series by using deep neural networks

Trabajo de integración curricular presentado como requisito para la obtención del título de Ingeniero en Tecnologías de la Información

Autor:

Oscar Vega

Tutor:

Oscar Chang. Ph.D.

Urcuquí, 1 de diciembre del 2021

Urcuquí, 1 de diciembre de 2021

SECRETARÍA GENERAL
(Vicerrectorado Académico/Cancillería)
ESCUELA DE CIENCIAS MATEMÁTICAS Y COMPUTACIONALES
CARRERA DE TECNOLOGÍAS DE LA INFORMACIÓN
ACTA DE DEFENSA No. UITEY-ITE-2021-00029-AD

A los 1 días del mes de diciembre de 2021, a las 10:00 horas, de manera virtual mediante videoconferencia, y ante el Tribunal Calificador, integrado por los docentes:

Presidente Tribunal de Defensa	Dr. IZA PAREDES, CRISTHIAN RENE , Ph.D.
Miembro No Tutor	Dr. MAYORGA ZAMBRANO, JUAN RICARDO , Ph.D.
Tutor	Dr. CHANG TORTOLERO, OSCAR GUILLERMO , Ph.D.

El(la) señor(ita) estudiante **VEGA ARELLANO, OSCAR ANDRES**, con cédula de identidad No. **1003618228**, de la **ESCUELA DE CIENCIAS MATEMÁTICAS Y COMPUTACIONALES**, de la Carrera de **TECNOLOGÍAS DE LA INFORMACIÓN**, aprobada por el Consejo de Educación Superior (CES), mediante Resolución **RPC-SO-43-No.496-2014**, realiza a través de videoconferencia, la sustentación de su trabajo de titulación denominado: **Forecasting time series using deep neural net**, previa a la obtención del título de **INGENIERO/A EN TECNOLOGÍAS DE LA INFORMACIÓN**.

El citado trabajo de titulación, fue debidamente aprobado por el(los) docente(s):

Tutor	Dr. CHANG TORTOLERO, OSCAR GUILLERMO , Ph.D.

Y recibió las observaciones de los otros miembros del Tribunal Calificador, las mismas que han sido incorporadas por el(la) estudiante.

Previamente cumplidos los requisitos legales y reglamentarios, el trabajo de titulación fue sustentado por el(la) estudiante y examinado por los miembros del Tribunal Calificador. Escuchada la sustentación del trabajo de titulación a través de videoconferencia, que integró la exposición de el(la) estudiante sobre el contenido de la misma y las preguntas formuladas por los miembros del Tribunal, se califica la sustentación del trabajo de titulación con las siguientes calificaciones:

Tipo	Docente	Calificación
Presidente Tribunal De Defensa	Dr. IZA PAREDES, CRISTHIAN RENE , Ph.D.	9,0
Tutor	Dr. CHANG TORTOLERO, OSCAR GUILLERMO , Ph.D.	10,0
Miembro Tribunal De Defensa	Dr. MAYORGA ZAMBRANO, JUAN RICARDO , Ph.D.	8,5

Lo que da un promedio de: **9.2 (Nueve punto Dos)**, sobre 10 (diez), equivalente a: **APROBADO**

Para constancia de lo actuado, firman los miembros del Tribunal Calificador, el/la estudiante y el/la secretario ad-hoc.

OSCAR ANDRES VEGA ARELLANO
Firmado digitalmente por OSCAR ANDRES VEGA ARELLANO
Fecha: 2021.12.01 13:33:23 -05'00'

VEGA ARELLANO, OSCAR ANDRES
Estudiante

CRISTHIAN RENE IZA PAREDES
Digitally signed by CRISTHIAN RENE IZA PAREDES
Date: 2021.12.01 13:37:59 -05'00'

Dr. IZA PAREDES, CRISTHIAN RENE , Ph.D.
Presidente Tribunal de Defensa

Firmado electrónicamente por:
OSCAR GUILLERMO CHANG TORTOLERO

Dr. CHANG TORTOLERO, OSCAR GUILLERMO , Ph.D.
Tutor

Firmado electrónicamente por:
JUAN RICARDO MAYORGA ZAMBRANO

Dr. MAYORGA ZAMBRANO, JUAN RICARDO , Ph.D.
Miembro No Tutor

DAYSY MARGARITA MEDINA BRITO

Firmado digitalmente por DAYSY MARGARITA MEDINA BRITO
Fecha: 2021.12.01 10:43:25 -05'00'

MEDINA BRITO, DAYSY MARGARITA
Secretario Ad-hoc

Autoría

Yo, **Oscar Andrés Vega Arellano**, con cédula de identidad **1003618228**, declaro que las ideas, juicios, valoraciones, interpretaciones, consultas bibliográficas, definiciones y conceptualizaciones expuestas en el presente trabajo; así cómo, los procedimientos y herramientas utilizadas en la investigación, son de absoluta responsabilidad de el autor del trabajo de integración curricular. Así mismo, me acojo a los reglamentos internos de la Universidad de Investigación de Tecnología Experimental Yachay.

Urcuquí, 1 de diciembre del 2021.

OSCAR ANDRES VEGA ARELLANO

Firmado digitalmente por OSCAR ANDRES VEGA ARELLANO
Fecha: 2022.01.04 17:01:22 -05'00'

Oscar Vega
CI: 1003618228

Autorización de publicación

Yo, **Oscar Andrés Vega Arellano**, con cédula de identidad **1003618228**, cedo a la Universidad de Tecnología Experimental Yachay, los derechos de publicación de la presente obra, sin que deba haber un reconocimiento económico por este concepto. Declaro además que el texto del presente trabajo de titulación no podrá ser cedido a ninguna empresa editorial para su publicación u otros fines, sin contar previamente con la autorización escrita de la Universidad.

Asimismo, autorizo a la Universidad que realice la digitalización y publicación de este trabajo de integración curricular en el repositorio virtual, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación Superior.

Urcuquí, 1 de diciembre del 2021.

OSCAR ANDRES VEGA ARELLANO

Firmado digitalmente por OSCAR ANDRES VEGA ARELLANO
Fecha: 2022.01.04 17:01:53 -05'00'

Oscar Vega
CI: 1003618228

Dedication

A mi Madre, el pilar de mi vida, quien ilumina mi camino desde las estrellas.

Acknowledgments

Special thanks to the University and its Professors for all the knowledge they shared in class.

Abstract

Forecasting and time series analysis plays a crucial role in many applications such as complex networks, demand forecasting, financial predictions, patient response to prescribed drugs, etc. In general, time series data behaves as a highly nonlinear process because of its stochastic nature; for instance, forecasting exports index data is very challenging, since this data is observed daily, weekly or even monthly. Recently improved prediction techniques based on Artificial Intelligence (AI) and Deep Neural Networks (DNNs) have appeared demonstrating excellent accuracy.

This work explores the combination of several DNNs techniques such as Multilayer Perceptron (MLP), Long Short-Term Memory (LSTM) and Convolutional Neural Network (CNN) for time series data prediction by using the concept of sliding window. The used dataset correponds to the Ecuadorian banana exports; it is stored in the Central Bank of Ecuador database and contains a register of the monthly banana exports over a period of 20 years. Experiments were carried out using Python 3 with Tensorflow & Keras, which makes possible to combine the capacity of inference of these networks. The results are analyzed by comparing the number of trainable parameters, sliding window size and root mean squared error (RMSE) values between six models: three sequential models such as MLP, LSTM and CNN, and three non-sequential models that combine MLP, LSTM and CNN in a parallel structure. The main results of this research show that by combining two DNNs in parallel, the computational complexity is decreased, and the capacity of inference and prediction in time series forecasting is improved.

Keywords: Time Series Forecasting, Deep Neural Networks (DNNs), Multilayer Perceptron (MLP), Long Short-Term Memory (LSTM) and Convolutional Neural Network (CNN)Sliding Window.

Resumen

El análisis de predicciones y series de tiempo juegan un papel importante en muchas aplicaciones como redes complejas, predicción de la demanda, financieras, respuesta de pacientes a medicamentos recetados, etc. En general, los datos de series de tiempo se comportan como un proceso altamente no lineal a causa de su naturaleza estocástica; por ejemplo, predecir los datos de índices de exportaciones es muy desafiante ya que estos datos son registrados diariamente, semanalmente o incluso mensualmente. Recientemente varias técnicas de predicción mejoradas basadas en Inteligencia Artificial y Redes Neuronales Profundas han aparecido demostrando una excelente precisión.

Este trabajo explora la combinación de varias técnicas de Redes Neuronales Profundas (DNN en inglés) como Perceptrón Multicapa (MLP en inglés), Memoria de Largo Plazo (LSTM en inglés) and Redes Neuronales Convolucionales (CNN en inglés) para la predicción de series de tiempo con el uso de la técnica de ventana deslizante. El conjunto de datos utilizado pertenece a las exportaciones de banano ecuatoriano, el cual está almacenado en la base de datos del Banco Central del Ecuador, y contiene el registro mensual de las exportaciones de banano hechas durante un periodo de 20 años. Los experimentos fueron llevados a cabo en Python 3 con Tensorflow & Keras, los cuales nos permiten combinar la capacidad de inferencia de estas redes. Finalmente, los resultados son analizados al comparar el número de parámetros por entrenar, tamaño de ventana deslizante y valores de la raíz del error cuadrático medio (RMSE en inglés) entre seis modelos: tres modelos sequenciales como MLP, LSTM y CNN, y tres modelos no secuenciales que combinan MLP, LSTM y CNN en una estructura paralela. Los principales resultados de esta investigación muestran que al combinar dos DNN en una estructura paralela la complejidad computacional disminuye y la capacidad de inferencia e predicción en la tarea de predicción de series de tiempo mejora.

Palabras Clave: Predición de Series de Tiempo, Redes Neuronales Profundas (DNNs), Perceptron Multicapa (MLP), Memoria de Largo Plaza (LSTM), Redes Neuronales Convolucionales (CNN), Ventana deslizante.

Contents

List of Tables

List of Figures

Listings

Chapter 1

Introduction

1.1 Background

Forecasting and time series analysis are relevant topics because they can help to explore and analyze different characteristics of data related to several fields such as traffic, finance, engineering, complex networks, and so on, [4]. The aim of time series analysis is to study the path observations of time series and build a model to describe the structure of data and then predict the future values of time series, [5]. For example, predicting projects' costs in order to optimize prices and schedule projects.

There are several traditional time series methods which have been proposed for time series predictions. For instance, stochastic methods, support vector machines (SVM), autoregressive integrated moving average (ARIMA), exponetial smoothing, etc., have successfully solved different time series forecasting problems, [4]. However, these approaches are not easily scalable to large datasets, [6], and require that the data follow a certain distribution which makes them unable to learn complicated non-linear patterns and relationships, [7].

Rencently, deep neural networks (DNNs) techniques have received an increasing amount of attention in time series analysis. Unlike traditional forecasting models, DNNs have a strong capability to identify complex patterns within and across non-linear relationships, extract higher order features, and can do so from datasets of raw time series with considerably less human effort, [8]. Moreover, studies that use DNNs for time series have shown great results and provide an extensive vision in the use of several DNNs architectures for time series analysis, [9].

1.2 Problem statement

The analysis of time series data that show the development of external trade, such as exports, plays an important role in a country's economy. For instance, in the last 25 years, Ecuador has been the leading exporter of bananas in the world. Actually, its banana trade represents for the country, after oil, the second source of income for its economy, [10]. Moreover, its exports have undergone a huge variation. According to [11], the exportation of this fruit until October 2020 amounts to 260.6 million of boxes, which represents 8.5% more than the exportation made in the same period of 2019 and higher figure in comparison with the figures reached 2016, 2017 and 2018.

On the one hand, DNNs techniques have caught the attention of many economic agents which are interesting in export forecast because these approaches provide valuable information and can help economic policy makers, monetary authorities, and exporting firms in making decisions, [12]. On the other hand, during the last years several techniques such as Multi-Layer Perceptron (MLP), Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM) model, among others, have been developed for predicting time series data.

This work aims to explore and combine several deep neural networks based methods such as MLP, CNN, and LSTM in order to know if by combining these networks, in a parallel structure, it is possible to improve the capacity of inference and learning of a single network. Moreover, to know which one of these offers best predictions regarding lower forecast errors and higher accuracy of forecasts, its going to be apply a specific dataset of the number of monthly banana exports, measure in metric tons and thousands of united states dollar (USD) first on board (FOB), made by Ecuador between the years 2000-2020.

1.3 Objectives

1.3.1 General objective

- To propose an exports prediction method based on the combination of deep neural networks.

1.3.2 Specific objectives

- To implement several deep neural networks from a combination of MLP, LSTM and CNN to do the predictions.
- To compare and analyze the performance and accuracy of all deep neural networks.
- To select the most accurate model based on metrics such as root mean squared error (RMSE) and loss value.

Chapter 2

Theoretical framework

This chapter gives an overview of the theoretical foundations of time series analysis and artificial neural networks. In the context of computational science, the following questions are answered:

- What is a time series?
- What is an artificial neural network?
- What is a deep neural network?

In order to answer the previous questions, this chapter starts with the theoretical foundation of time series and its main components. Then, the definition of artificial neural networks and its main components are presented. Finally, the concept of deep neural networks, optimizer and three of the most popular models for time series forecasting are presented.

2.1 Time series

A time series is a set of observations measured sequentially through time (every hour, day, week, month, year, or any other regular interval). Depending on the nature of analysis and practical needs, there can be applied different kinds of time series, [13]. Typically a time series (see Figure 4.2) is represented by a graph where the observations are plotted against corresponding time in order to visualize and analyze the pattern of the data. Some examples are: the continuous monitoring of a person's heart rate, hourly readings of temperature, daily sales price of a company stock, monthly imports-exports of any particular product in a country, among several others.

2.2 Components of a time series

Usually a time series is composed and affected by four main components: Trend, Cyclical, Seasonal and Irregular components (see Figure 2.1).

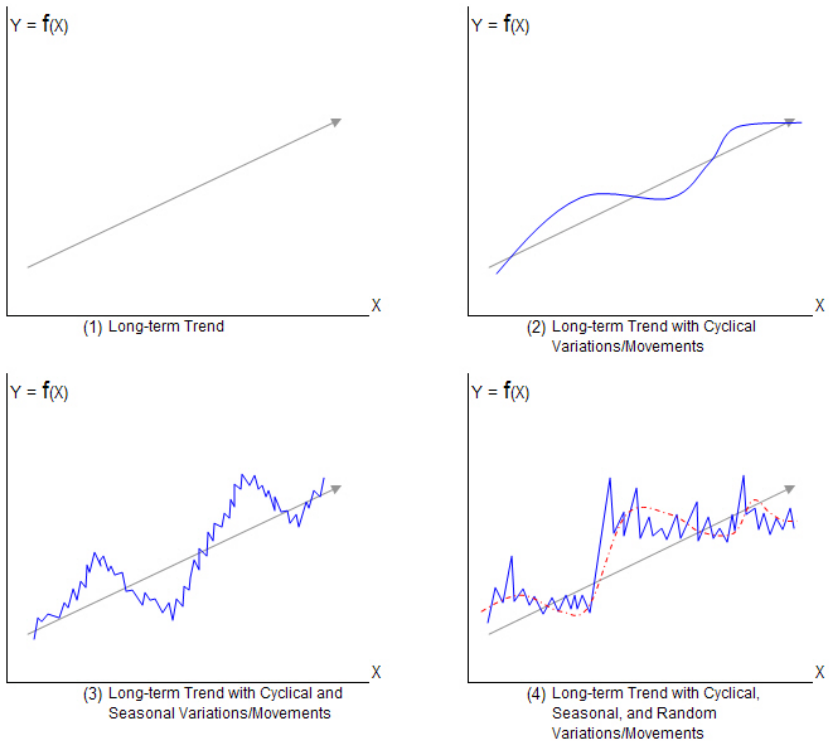

Figure 2.1: Components of a time series. Source: [1].

2.2.1 Trend

The tendency of a time series to increase, decrease or stagnate over a long period of time is named as Secular Trend or Trend. In other words, that trend is the long term direction in a time series, [13]. For example, time series data relating to population growth, number of houses in a country, etc., show an increasing trend, whereas a decreasing trend can be visualized in series relating to mortality rates, quantity of available natural resources, etc.

2.2.2 Seasonal

Seasonal variations in a time series are fluctuations within a year during a season, [13]. Some relevant factors that produce seasonal variations are: weather and climate conditions, customs, etc. For example, sales of ice-cream increase during summer, sales of woolen cloths increase during winter. Seasonal variation is a relevant component for businessmen, shopkeeper and producers because it help to organize and make proper future plans, [13].

2.2.3 Cyclical

The cyclical variations are the oscillatory movements in a time series and details the medium-term changes which repeat in cycles. The duration of a cycle extends over around two or more years. Most of the economic and financial time series show some kind of cyclical variation, [13]. For example a business cycle consists of four phases: Prosperity, Decline, Depression and Recovery.

2.2.4 Irregular

Unpredictable factors, which are not regular and also do not repeat in a particular pattern, produce the irregular or random variations in a time series. These variations are caused by events such as a rise in prices of steel due to strike in the factory, accident due to failure of the break, flood, war, strike, earthquake, revolution, etc. There is no statistical technique for measuring random fluctuations in a time series, [13].

2.3 Artificial neural networks (ANNs)

ANNs are computerized models of a human brain designed to simulate the way in which human brain processes information, [2]. For instance, ANNs are trained through experience with appropriate learning examples (data) just as people do, not from programming, [2]. So, the main advantage of ANNs over other computational models is that they can learn to solve problems by itself, [14]. Some other advantages include:

- **Adaptive learning:** An ability to learn how to do tasks based on the data given for training or initial experience, [15].
- **Self-Organisation:** ANNs can create its own organisation or representation of the information it receives during learning time, [15].
- **Real Time Operation:** ANNs computations may be carried out in parallel, that is why during the last years hardware devices have been designed and manufactured in order to take advantage of this capability, [15].

2.4 ANNs components

There are many types of neural networks designed for several purposes, [2]; however, most of them can be described by the neurons, weights, neurons layers, and transfer functions of their neurons (see Figure 2.2).

2.4.1 Neurons

The artificial neuron is the component designed to simulate the function of the biological neuron. The inputs or arriving signals, multiplied by the connection weights (adjusted) are first summed (combined) and then passed through a activation function to produce the output for that neuron, [2].

2.4.2 Neurons layers

An ANN is a system of fully interconnected neurons which it is organized in three types of layers: the input layer, the output layer, and the hidden layers between them. The input layer neurons receive data from a data file, the output layer neurons provides the final result or response to the input data, and the hidden layers neurons is the responsible for communicating with other neurons. So, they are part of the large internal pattern that determines a solution to the problem, [2], and process the data. Theory says that most functions can be approximated using a single hidden layer, [16].

2.4.3 Transfer or activation function

An ANN use the transfer or activation functions to transform an input signal into an output signal which in turn is the input parameter to the next layer in the network, [17]. Therefore, the activation function is the weighed sum of the neuron's inputs and it may be anything like sigmoid, hyperbolic, tangent functions, among others, [18].

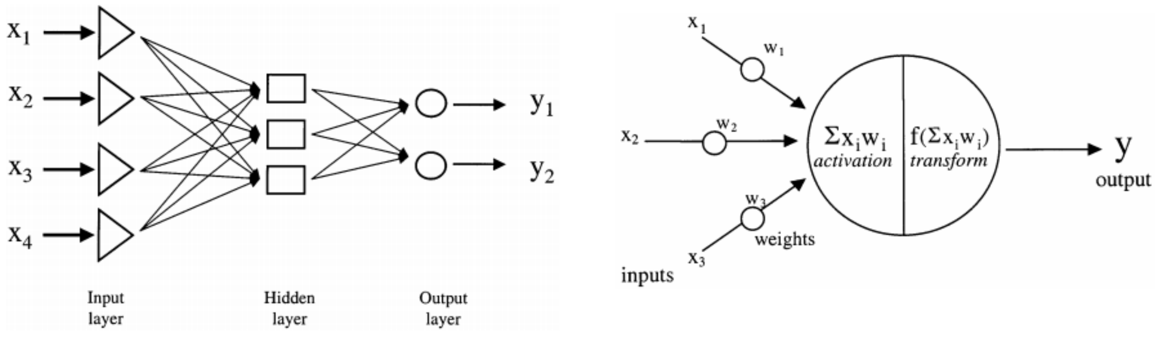

Figure 2.2: Artificial neural network and neuron model, respectively. Source: [2].

2.5 Deep neural networks (DNNs)

A Deep neural network (DNN) is a multilayer techniques that has gained popularity among researchers. DNNs are characterized by having more than one layer of hidden units among its inputs and its outputs, [19]. In recent years, DNNs have received an increasing amount of attention in time series analysis due to its capability of mapping highly nonlinear input-output data samples, [20], to extract higher order features, and identify complex patterns within and across it, [8].

2.5.1 Multilayer Perceptron

Multilayer Perceptron (MLP) is a type of DNN that consists of input layer, output layer and hidden layers between these two layers. In these layers it's found a system of simple interconnected neurons, or nodes associated to weights, which is a model that represents a nonlinear mapping between an input and an output, [21]. The number of these layers depends on the problem, [22]; therefore, optimizing the number of hidden layers and the number of neurons in each hidden layer can increase the speed and efficiency of the neural network, [23].

MLP is one of the most popular models in time series forecasting which is predominantly trained using the Backpropagation (BP) algorithm, [24]. The BP algorithm uses the gradient descent technique where it is expected that the weights will converge to the global minimum of the error surface, [21]. On the one hand, the BP algorithm may be implemented as an on-line training where the network weights are adjusted after each pattern has been presented, [21]. On the other hand, it may be implemented as an batch training, where the summed error for all patterns is used to update the weights, [21].

2.5.2 Long Short-Term Memory

Long Short-Term Memory (LSTM) was designed as a specific type of recurrent neural network (RNN) in order to model temporal sequences. LSTM was initially introduced by Hochreiter and Schmidhuber (1997), and the main objectives of LSTM are to model long-term dependencies and determine the optimal time lag, for time series problems, more accurately than the traditional RNN, [25].

LSTM is a type of DNN that consist of one input layer, one output layer, and recurrent hidden layers between these two layers. Unlike traditional ANNs, the basic unit of the hidden layers are the memory blocks, [26]. A memory block has four main elements: the memory cell, and the three logistic gates. The memory block holds information and the logistic gates specify the flow of data inside the LSTM, [27]. The input gate writes and update states into the memory cell, the output gate decides what the next hidden state should be, and the forget gate controls what information to throw away from memory and decides how much of the past it should remember.

The LSTM gates are similar to the artificial neurons in an ANN since they are multiplicative analog sigmoid-activated nodes, [27]. Moreover, the process of manipulating the states trough these gates is relevant because it helps to the recurrent network to remember what it needs, to forget what is no longer useful and lastly to solve vanishing gradient and exploding gradients problems, [27].

2.5.3 Convolutional Neural Network

Convolutional Neural Networks (CNNs) are similar to traditional ANNs since they are comprised of neurons organized in three dimensions: the spatial dimensionality of the input (height and the width) and the depth, which self-optimise through learning, [28]. Moreover, these neurons reside into three types of layers (convolutional layers, pooling layers and fully-connected layers) which are part of the CNNs as well.

CNNs are a type of DNN which has recently gained popularity due to its success in time series problems, [29]. Unlike traditional ANNs, the output of any given CNN layer is connected only to local regions in the input. This is achieved by sliding a filter or weight matrix over the input, and at each point it is computed the dot product between the two; in other words, it is computed a convolution between the input and filter. Hence, this structure has allowed to the model to recognize specific patterns in the input data by learning through filters, [30].

2.6 Optimizer

Training very large deep neural networks can involve a lot of time. There are four ways to speed up the training and reach a better solutions: applying a good initialization strategy for the connection weights, using a good activation function, using Batch Normalization, and using a faster optimizer than the regular gradient descent optimizer, [3].

Adam optimization

Adam stands for adaptive moment estimation. It is an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iterative based in training data, [31]. Adam combines the ideas of momentum optimization and root mean squared propagation (RMSProp): just like momentum optimization it keeps track of an exponentially decaying average of past gradients (steps 2.1), and just like RMSProp it keeps track of an exponentially decaying average of past squared gradients (steps 2.2).

$$\mathbf{m} \leftarrow \beta_1 \mathbf{m} - (1-\beta_1)\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \tag{2.1}$$

$$\mathbf{s} \leftarrow \beta_2 \mathbf{s} + (1-\beta_2)\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \otimes \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \tag{2.2}$$

$$\widehat{\mathbf{m}} \leftarrow \frac{\mathbf{m}}{1-{\beta_1}^t} \tag{2.3}$$

$$\widehat{\mathbf{s}} \leftarrow \frac{\mathbf{s}}{1-{\beta_2}^t} \tag{2.4}$$

$$\theta \leftarrow \theta + \eta \widehat{\mathbf{m}} \oslash \sqrt{\widehat{\mathbf{s}} + \varepsilon} \tag{2.5}$$

Steps 2.3 and 2.4 are a technical detail: since $\mathbf{m}$ and $\mathbf{s}$ are initialized at 0, they will be biased toward 0 at the beginning of training, so these two steps will help boost m and s at the beginning of training, [3]. $J(\boldsymbol{\theta})$ represents the cost function of the gradient descent with regards to the weights $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$. A detailed information about gradient descent and its main variants can be found in [3].

Adam configuration parameters:

In [32] it is suggested that good values for testing machine learning problems may be: $\eta = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\varepsilon = 10^{-8}$.

- η**:** The learning rate is the proportion in which the weights are updated. Larger values (e.g. 0.3) results in a faster initial learning before the rate is updated. Smaller values (e.g. 1.0^{-5}) slow down the learning during training.
- β_1**:** The exponential decay rate for the first moment estimates.
- β_2**:** The exponential decay rate for the second moment estimates.
- ε**:** Is a very small number to prevent any division by zero in the implementation.

2.7 Building complex DNNs

In this section, the software and tools used to assembled different types of DNNs are described. The following questions are answered over the course of this section:

- What is Tensorflow & Keras?
- How to assemble sequential and non-sequential DNN?

In addition, several structures of DNNs with code and graph examples are explained.

2.7.1 Tensorflow & Keras

Tensorflow is an open source platform for distributed numerical computations. It was designed to distribute the computations across potentially hundreds of multi-GPU servers in order to train and execute very large neural networks efficiently, [3]. Moreover, TensorFlow was developed at Google as a flexible ecosystem of libraries, tools, and community resources to support machine learning and deep neural networks research; in fact, the system can be applied to a wide variety of other domains, [33].

Keras is a high level Deep Learning API designed for solving machine learning problems. It can run on top of either TensorFlow, Theano or Microsoft Cognitive Toolkit (CNTK); for instance, TensorFlow has its own implementation of this API which is named tf.keras and it provides support for several advanced TensorFlow features such as to assemble, fit, train and evaluate neural networks, [3].

Tensorflow & Keras offers two ways to build neural networks. On the one hand, the Sequential API is quite easy to use for assembling sequential models which are topologies extremely common, [3]. On the other hand, it is sometimes useful to assemble neural networks with more complex topologies, or with multiple inputs or outputs, [3]; therefore, Tensorflow & Keras offers the functional application programming interface (API) for this type of non-sequential models.

2.7.2 Non-sequential and sequential models

One example of a non-sequential neural network is a Wide & Deep neural network. This type of neural network was introduced in [34], where its structure connects all or part of the inputs directly to the output layer (see Figure 2.3). In fact, due to this architecture the neural network can learn both simple rules (through the short path) and deep patterns (using the deep path), [3]. On the contrary, a common neural network such as MLP may distorted, by this sequence of transformations, the simple patterns in the data due to it forces to all the data to flow through the full stack of layers, [3].

Example 1 of non-sequential model

In this first example, the code shows how to assemble the Wide & Deep neural network depicted on Figure 2.3:

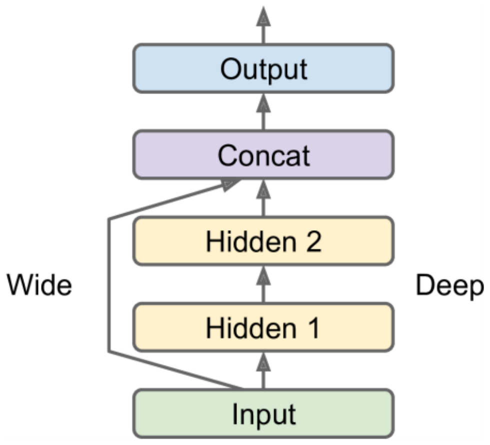

Figure 2.3: Wide & deep neural network. Source: [3].

Listing 2.1: Python code to assemble a Wide & Deep neural network

```python
input = Input(shape=(12, 1))
hidden_1 = Dense(7, activation='relu')(input)
hidden_2 = Dense(7, activation='relu')(hidden_1)
concat = Concatenate([input, hidden_2])
output = Dense(1)(concat)
model = Model(inputs=input, outputs=output)
model.compile(loss='mean_squared_error', optimizer='adam', metrics
    =[rmse])
plot_model(model, to_file='Wide&Deep', show_shapes=True)
model.summary()
```

Analysis of each line:

- Line 1: it creates an Input object, named input, that contains the inputs parameters for the network.
- Line 2: it creates a Dense layer, named hidden_1, with 7 neurons and the ReLU activation function. Once the Dense layer is created, notice that hidden_1 acts as function and input is called like a function, as well. That is why this is called functional API. Note that this initial lines are just telling to Keras how it should connect the layers together and no actual data is being processed yet. It is necessary to point out a Dense layer in Keras it is considered an usual layer which can contain several neurons and in this case acts such as a hidden layer.
- Line 3: it creates a second hidden layer which it is used as a function. Note that the output of the first hidden layer pass as input for the second hidden layer.
- Line 4: it this is the key step to build the wide path. Here a concatenate layer is created in order to use it as a function and for concatenating the input (wide path) and the output (deep path) of the second hidden layer. Note: to build a sequential

model this line needs to be deleted and hidden_2 may pass as input parameter to the output layer.

- Line 5: it creates the output layer with a single neuron and no activation function. Once again notice that the output is called like a function since it is passed as input, the result of the concatenation.
- Line 6: it creates the Keras Model by specifying which inputs and outputs to use.
- Line 7: once it is assembled the Keras model, it is necessary to compile the model, train it, evaluate it and use it to make predictions. Here the loss, optimizer and metrics are: mean squared error, Adam and root mean squared error, respectively
- Line 8: the plot_model saves a graph of the neural network. In this example, the image is save as a file named ”Wide&Deep”
- Line 9: the model.summary displays the relevant information about the structure of the network such as number of layers, neurons, weights, etc.

Example 2 of non-sequential model

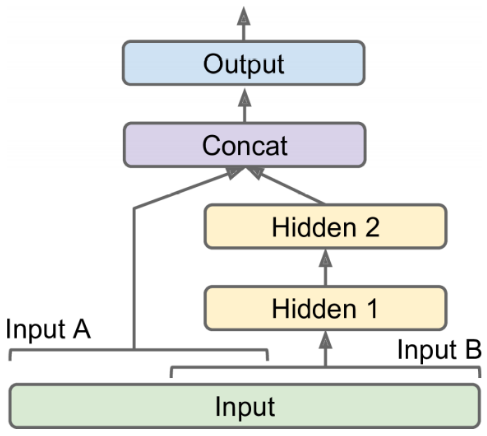

Figure 2.4: Wide & deep neural network with multiple inputs. Source: [3].

The following code sends a subset of the input features through the wide path, and a different subset (overlapped) through the deep path. One solution for this problem is to use multiple inputs. For instance, send 5 features through the deep path (features 0 to 4), and 6 features through the wide path (features 2 to 7) as on Figure 2.4:

Listing 2.2: Python code to assemble a Wide & Deep neural network with multiple inputs

```
1    input_A = Input(shape=(5, 1))
2    input_B = Input(shape=(6, 1))
3    hidden_1 = Dense(7, activation='relu')(input_B)
4    hidden_2 = Dense(7, activation='relu')(hidden_1)
5    concat = Concatenate([input_A, hidden_2])
```

```
6    output = Dense(1)(concat)
7    model = Model(inputs=[input_A, input_B], outputs=output)
8    model.compile(loss='mean_squared_error', optimizer='adam', metrics=[rmse])
9    plot_model(model, to_file='Wide&Deep', show_shapes=True)
10   model.summary()
```

Analysis of each line:

- The code is exactly the same as the above example. However, the line 1 and 2 have a different input shape since it pass different sections of the same data. Moreover, in the line 7 it is also necessary specify that there are two inputs instead of one as on the previous example.

Example 3 of non-sequential model

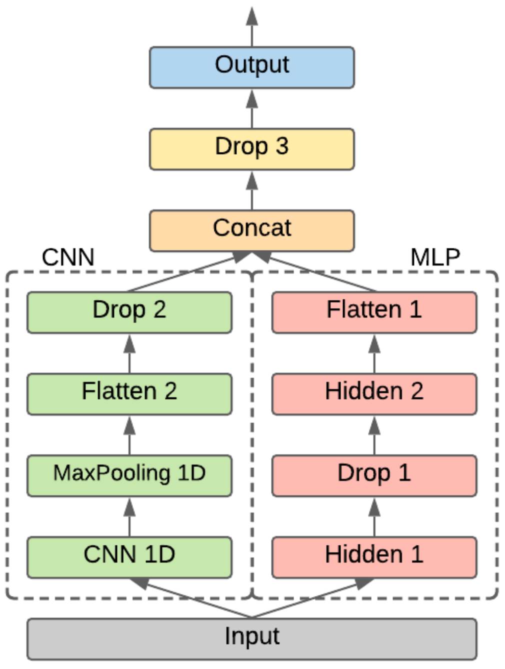

Figure 2.5: Parallel deep neural network with multiple networks.

Finally, this code uses the wide path to assemble a CNN and combine it with another different neural network such as MLP as on Figure 2.5:

Listing 2.3: Python code to combine CNN & MLP

```
1    input = Input(shape=(12, 1))
2    hidden_1 = Dense(7, activation='relu')(input)
3    drop_1 = Dropout(0.1)(hidden_1)
4    hidden_2 = Dense(7, activation='relu')(drop_1)
```

```
5     flatten_1 = Flatten()(hidden_2)
6     cnn1D = Conv1D(filters=4, kernel_size=7, strides=1, activation='
         relu', padding="same")(input)
7     mxPl1D = MaxPooling1D(pool_size=2)(cnn1D)
8     flatten_2 = Flatten()(mxPl1D)
9     drop_3 = Dropout(0.1)(flatten_2)
10    concatenated_1 = Concatenate([drop3,flatten_1])
11    drop_2 = Dropout(0.1)(concatenated_1)
12    output = Dense(1)(drop_2)
13    model = Model(inputs=input, outputs=output)
14    model.compile(loss='mean_squared_error', optimizer='adam', metrics
         =[rmse])
15    plot_model(model, to_file='CNN_MLP', show_shapes=True)
16    model.summary()
```

Analysis of each line:

- Line 1: it creates an Input object, named input, that contains the inputs parameters for MLP and CNN.
- Line 2: it creates a Dense layer, name hidden_1, with 7 neurons and the ReLU activation function.
- Line 3: it creates a Drop layer, named drop_1, which it is used as a function because it is passed the previous layer, dense1, as a function.
- Lines 2 to 5: they create several layers in order to assemble a MLP with 2 hidden layers and 1 drop layer between both layers.
- Line 6: it creates a Convolutional layer (1 dimension) with 4 filters, a kernel size of 7x1, size of the stride equal to 1, RelU activation and padding of "same". Notice that the purpose of this code is combine two different DNNs. So, since this is the first layer of the CNN, it is necessary to pass the inputs1 created in line 1, just as the same way that it is made for the first Dense layer of MLP in line 2.
- Line 7 to 9: they create a Max Pooling, Flatten and a Drop layer, respectively. This layers are passed as function to its respective next layers, just as it is made for MLP.
- Line 10: this is the key step to combine two networks. Here the Concatenate layer is created in order to concatenate the outputs of the two last layers of CNN and MLP (drop3 and flatten1, respectively and passed these as function).
- Line 11-12: they create a Drop and Dense layer with 1 neuron, where all the concatenated outputs will pass to finally obtain the result.
- Line 13-14: Once it is assembled the Keras model it is necessary to compile the model, train it, evaluate it and use it to make predictions. Here it is used as loss, optimizer and metrics: mean squared error, Adam and root mean squared error, respectively.

- This DNN is assembled to predict one month of banana exports, that is why there are one neuron at the last layer. The 12 inputs in the first layer correspond to the samples of the monthly banana exports.

Chapter 3

State of the art

Vasquez, [35], and Mayón, [10], show the importance of banana trade for Ecuador which becomes the second product more exported in the country and a relevant source of incomes for the development of its economy. Hence, in this work it will be explored the time series assembled from Ecuadorian banana trade regarding time and exports, by using a combination of several popular deep neural networks.

DNN's use several hidden layers as a learning technique, for mapping highly nonlinear input-output data, and have gained an increasing popularity in the time series analysis research, [20]. Horák, Šuleř & Vrbka, [36], show how most researches indicate that DNN's models are better than traditional statistical or mathematical models. This work is relevant because they compare the accuracy of time series by means of regression analysis and deep neural networks on the example of the USA exports to China. Moreover, they discuss the possible uses and advantages of deep neural networks in practice.

Chang *et al.*, [37], propose a DNN which explores a few years of weekly pharmaceutical sales data and learns to make assertive predictions. The authors assembled a time series from one variable (weekly pharmaceutical sales), normalized these data, and applied the sliding window method to obtain different learning scenarios to predict a "near future". It is important take into account this system because it produced a good hit rate of forecasting, where many values from the original time series data were well predicted.

LSTM is a subtype of DNN that is able to keep lots of information and it is more appropriate for modeling economic data where there is a long-term dependence in time series due to the lagged effect, [38]. Vidya & Prabheesh, [39], measured and predicted the future trade interconnectedness among countries before and after the COVID-19 outbreak. This study is relevant, since they propose an optimized LSTM model, with: one input layer, one output layer, and one hidden layer which store 50 LSTM cells, which minimized its MSE when it predicts the future values of exports and imports up to the end of the year 2020.

Naqvi *et al.*, [40], focused on developing an accurate wheat production forecasting model by using LSTM neural networks in conjunction with a data pre-processing smoothing mechanism. Its smoothing function works with a polynomial of degree two, and a window size of four years that is used to further improve the prediction accuracy. The importance of this paper is crucial since they show despite of having excellent forecasting capabilities, ANN and LSTM-NN have had limited application in the field of agriculture and crop production forecasting.

Convolutional Neural Network (CNN) is another subtype of DNN that has several full-connected layers. It uses this name from mathematical linear operation between matrixes named convolution and has excellent performance in processing two-dimensional data such as images and videos, [41]. Kouassi & Moodley, [42], used a hybrid deep neural network structure named TreNet that combines both a LSTM and a CNN, and takes in a combination of raw data points and trend lines such as its input for making predictions. Moreover, the authors suggest the importance of using an appropriate validation method for evaluating and developing machine learning models for trend prediction in time series data.

Borovykh, Bohte & Oosterlee, [30], study the performance of a CNN known as WaveNet. The network uses layers of dilated convolutions applied to the inputs in order for learning the trends and relations in and between the data. It is relevant to know that the authors proved that WaveNet model is a strong competitor to LSTM models, in particular when taking into consideration the training time. In addition, they conclude that even though time series forecasting remains a complex task and finding one model that fits all is hard, WaveNet is a simple, efficient and easily interpretable network that can act as a strong baseline for forecasting.

Li *et al.*, [43], proposed a hybrid deep learning model by combining wavelet packet decomposition (WPD) and long short-term memory (LSTM). The network is used to predict one-hour-ahead of photovoltaics (PV) power with five-minute intervals. First, WPD decompose the original PV power series into sub-series. Then, four independent LSTM networks are applied to the sub-series in order to predict the results. This study is relevant because they demonstrated that its proposed hybrid model shows a superior performance than LSTM, MLP, gate recurrent (GR) and recurrent neural networks (RNN).

Huang *et al.*, [44], assembled a hybrid deep neural model named WPD–CNN–LSTM-MLP for solar irradiance forecasting. The structure of this network is based on multi-branches with multi-variable inputs that includes hourly solar irradiance and three climate variables such as: temperature, relative humidity, and wind speed and their combination. This paper shows the importance of these new hybrid models; for example, WPD–CNN–LSTM-MLP is able to extracts the inherent characteristics of multi-layer inputs, overcomes the problems of traditional models, and achieves higher accurate forecasting results.

Chapter 4

Methodology

In this chapter the methodology applied to assemble and compare the DNNs structures, is described (see Figure 4.1). The following questions are answered in this chapter:

- Where the dataset comes from?
- How to normalize the data set?
- How to transforms the dataset into a supervised learning problem?
- What is the structure of the DNNs assembled?
- How does the training process for the sequential and non-sequential DNNs work?
- Which metrics will be used to compare the performance of each model?

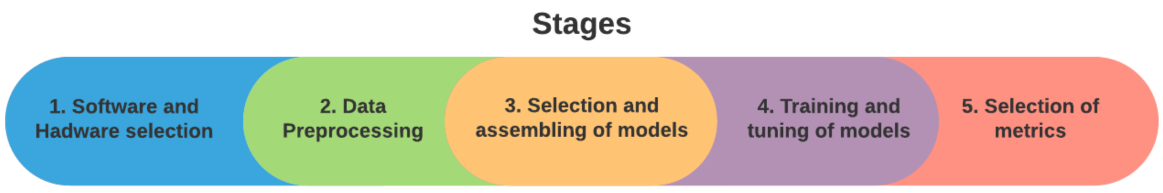
Stages
1. Software and Hadware selection
2. Data Preprocessing
3. Selection and assembling of models
4. Training and tuning of models
5. Selection of metrics

Figure 4.1: Methodology stages

4.1 Hardware and software

The entire project is developed on a Linux OS: Ubuntu 20.04. It is used Python 3.8.5 with the support of tools such as Tensorflow 2.3.1 & Keras 2.4.3 that is described in section 2.7.

4.2 Dataset

The two datasets analyzed in this project are available on the website of Central Bank of Ecuador[1], which it is updated each month. Both datasets contain 248 values which represent the quantity of monthly banana exports, measured in metric tons and thousands of USD FOB, made by Ecuador from January 2000 to August 2020. All this data is stored in a .csv file and it is divided into 75% for training, 5% for validation and 20% for testing.

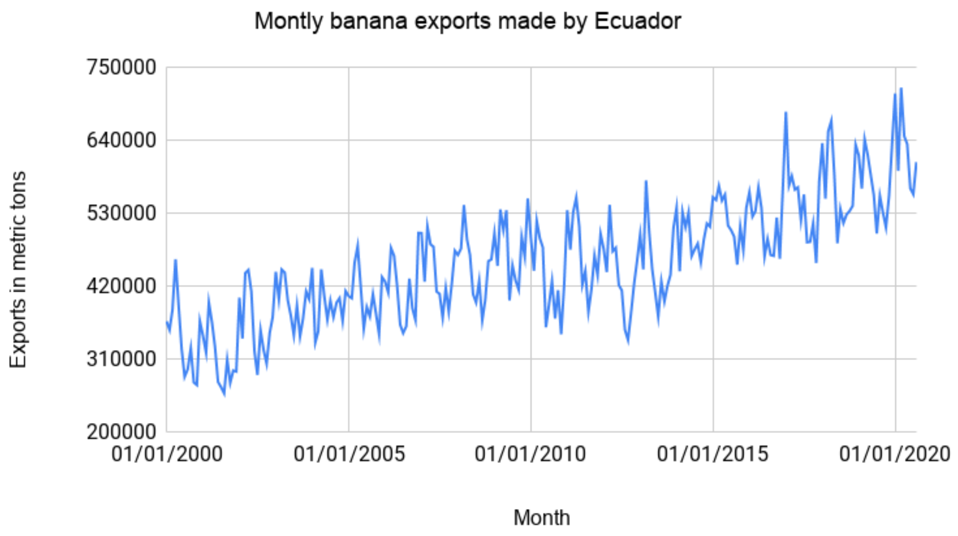

Figure 4.2: Time series

4.2.1 Data normalization

The effectiveness of any learning algorithm is heavily dependent on the normalization method. The time series data can have a wide range of values, so it need to be scaled to a common range of values to speed up the learning process, [45]. An adequate normalization previous to the training process of the DNN allows the model to be less sensitive to the scale of features, to reduce significantly calculation time, to converge to better weights and, in turn, leads to improve the accuracy of the model, [46].

There are several data normalization techniques available. In this case it is used the Min-Max normalization, where all the data is scaled to a range between [0, 1] by using the next formula:

$$x_{\text{norm}} = \frac{(x_i - \min(x))}{\max(x) - \min(x)} \tag{4.1}$$

Where:

- x_i: is the the original value, of the time series, at the time i

[1] https://www.bce.fin.ec/index.php/informacioneconomica/sector-externo

- x_{norm}: is the normalized value of the time series
- $\min(x)$: is the minimum value of the time series
- $\max(x)$: is the maximum value of the time series

4.2.2 Data as supervised learning

The sliding window method is one of the several techniques to convert the data into a supervised learning problem. Once the data pass by the process of normalization, this method place the values of the time series data in temporary segments, [20]. For instance, after selecting the first segment, the next segment is selected from the end of the first segment; then, the process is repeated until all time series data are segmented. In Figure 4.3 it is appreciated a graphical representation of the sliding window method and how it use the time series data as inputs and outputs to feed a neural network.

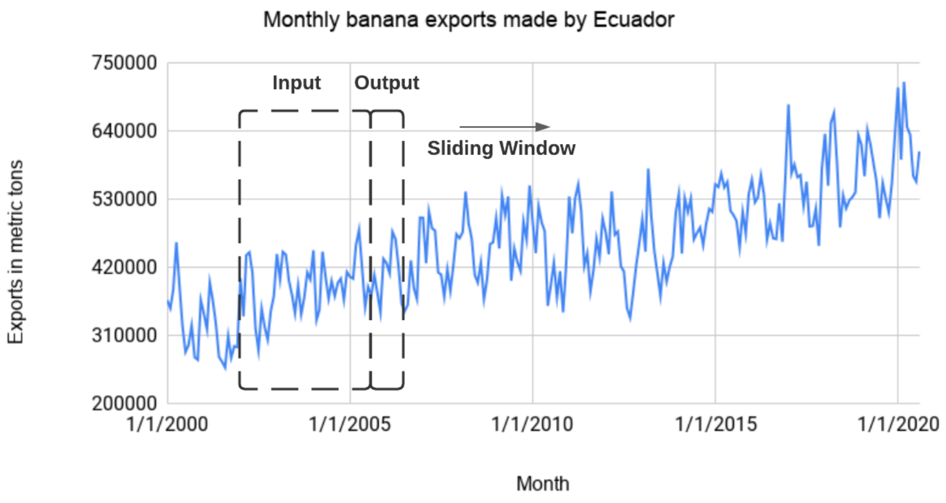

Figure 4.3: Sliding window process.

According to [20], the size of the window and segment can increase until the less error approximation is reached. Therefore, in this project is used a window size of 6, 12 and 24 values. It means that we move around the values taking 6, 12 and 24 moths as inputs and 1 month as output.

4.3 DNNs: Sequential models for forecasting

In this section the structure of the three most common DNNs, such as MLP [24], LSTM [25] and CNN [29], found in literature to make time series predictions, it is defined. It is important to mention that in the testing phase, once a new export value is predicted, then that value is also used to do the new predictions.

4.3.1 MLP

MLP is the first deep neural network considered. Its structure is similar to the one presented in [37]; then, from this study are extracted the most important parameters such as the number of neurons, learning coefficient, loss function, etc. Moreover, a few other parameters are calculated experimentally to obtain the expected results of this project. Table 4.1 shows the final parameters setup for this model and Figure 4.4 depicts the structure for a MLP with 6 inputs.

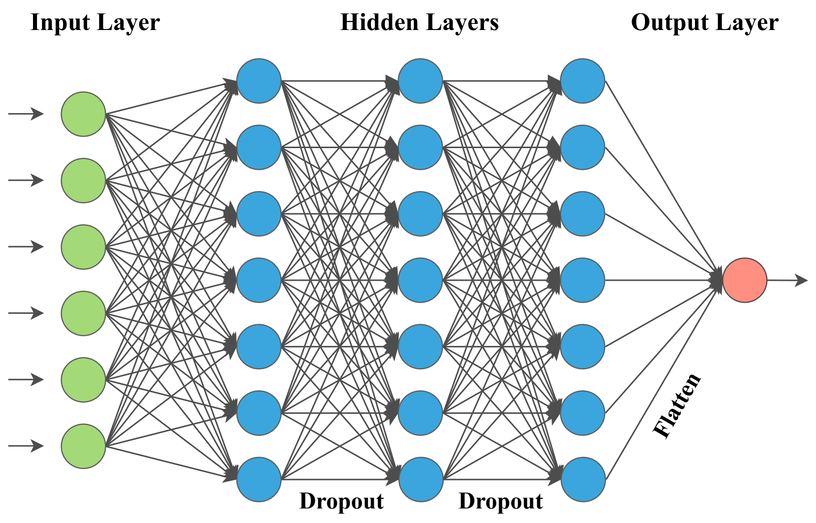

Figure 4.4: Graph for MLP.

Hiperparameters	#
Input layer	6,12,24
Neurons on input layer	1
Hidden layer(s)	3
Neurons by hidden layer	7
Activation function	ReLU
Flatten Layer	1
Optimizer	"Adam"
Loos function	Root Mean Squared Error
Dropout layers	3
Dropout value	0.1
Mini-batch size	32
Epochs	700

Table 4.1: Hiperparameters for MLP

The process of training for MLP works in the following way:

- Once the monthly exports dataset has been normalized and reshaped as a supervised learning problem, the input layer receives the information through the 6, 12 and 24 inputs neurons(this number depends on the size of the sliding window).
- Next, the inputs neurons propagate this information, in a sequential way, to the next two hidden layers which are composed of 7 neurons. These neurons are in charge of computing a weighted sum between their respective weights and the data coming from the previous hidden layers. After that, hidden neurons apply the ReLU activation function to the weighted sum obtained in each hidden layer.
- Now, the output layer which in this case has only one neuron, receives the 7 outputs of hidden layer, and performs a weighted sum and applies again the RELU activation function in order to obtain the final output. This output corresponds to the export prediction one month ahead after the input data.
- Finally, Once the output is computed for one complete batch, in this case a batch size of 21, the DNN computes the RMSE loos function by comparing actual output to the desired (target) output in order to adjust weights and minimise the overall error by using the "Adam" optimizer and BackPropagation.
- One relevant point to keep in mind is that the only purpose of the Flatten layer is to reshape the size of the data that pass thought it. For example, if flatten is applied to a layer which have a input shape as (batch_size, 2,2), then the output shape of the layer will be (batch_size, 4). Moreover, the purpose of the Dropout layer is randomly sets input units to 0 with a frequency of rate at each step during training time, which helps to prevent overfitting, [47].

4.3.2 LSTM

LSTM is the second deep neural network considered. Its structure is similar to the one presented in [27]; then, from this study are extracted the most important parameters such as the number of neurons, learning coefficient, loss function, etc. Moreover, a few other parameters are calculated experimentally to obtain the expected results of this project. Table 4.2 shows the final parameters setup for this model, and Figure 4.4 depicts the structure for a LSTM with 6 inputs as data.

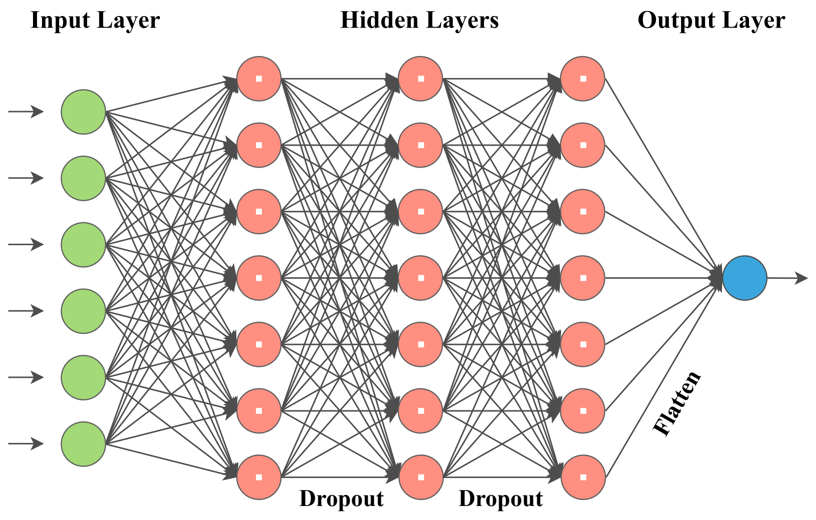

Figure 4.5: Graph for LSTM.

Hiperparameters	#
Input layer	6,12,24
Neurons on onput layer	1
Hidden layer(s)	3
Neurons by hidden layer	7
Activation function	ReLU
Flatten Layer	1
Optimizer	“Adam”
Loos function	Root Mean Squared Error
Dropout layers	3
Dropout value	0.1
Mini-batch size	32
Epochs	700

Table 4.2: Hiperparameters for LSTM

The process of training for LSTM works in the following way:

- This process works in the same way such as MLP; however, LSTM is composed of three additional logic gates ,within a cell (or neuron), which have also an associated weight. Unlike other neural networks, a LSTM performs several operations (not just weigted sum) within the cells to obtain the output. These operations such as sigmoid activation, tangential activation, pointwise multiplication, pointwise addition and vector concatenation allows to this type of DNN to keep or forget the information, [48].

4.3.3 CNN

CNN is the third deep neural network considered. Its structure is similar to the one presented in [37]; then, from this study are extracted the most important parameters such as the number of neurons, learning coefficient, loss function, etc. Moreover, a few other parameters are calculated experimentally to obtain the expected results of this project. Table 4.3 shows the final parameters setup for this model, and Figure 4.6 depicts the structure for a CNN with 6 inputs as data.

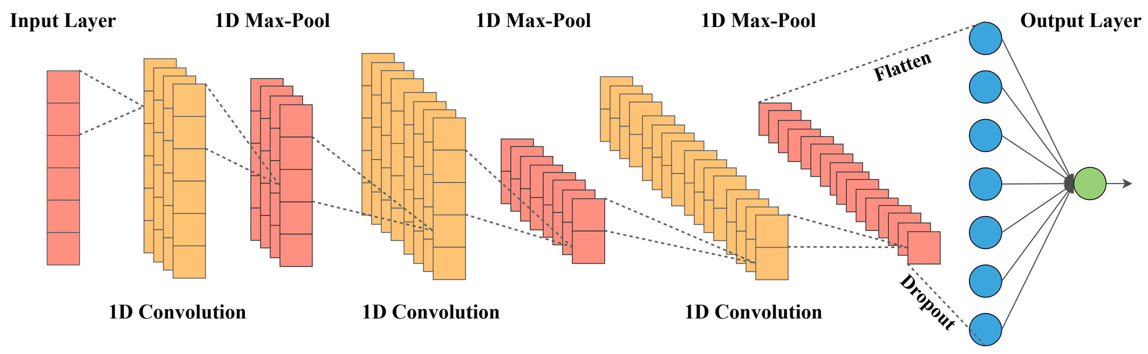

Figure 4.6: Graph for CNN

Hiperparameter	#
Input layer	6,12,24
Conv1D layer	3
Filters by Conv1D	4,8,16
Kernel size by Conv1D	7,5,3
Stride size by Conv1D	1
Max-Pooling layer	3
Pool size by Max-Pol	2
Stride size by Max-Pol	1
Activation	ReLU
Optimizer	“Adam”
Loos function	Root Mean Squared Error
Dropout layer	3
Dropout value	0.1
Mini-batch size	32
Epochs	700

Table 4.3: Hiperparameters for CNN

The process of training for CNN works in the following way:

- This process works in the same way such as MLP; however, it has additional convolutional layers. These convolutional layers replace the fully connected layers, and therefore reducing the amount of weights in its structure, [49]. Unlike other neural networks, a CNN performs several operations (not just weigted sum) with filters of different sizes in order obtain the output. These operations usually are the convolutional product between the convolutional layers and the filters.

4.4 DNNs: Non-sequential models for forecasting

In this section, it is defined the structure of the three non-sequential DNNs models proposed in this project to make time series predictions. It is important to mention that the structure of this networks use the wide path to assemble another DNN in order to combine two different models such as shown in Figure 2.5. Moreover, during the testing phase all these networks use the previously predicted values to predict the new exports values.

4.4.1 MLP&LSTM

MLP and LSTM are combined to assemble this fourth neural network. Its structure and hiperparameters are the same as the described on Table 4.1 and Table 4.2 for MLP and LSTM, respectively; however, it is decreased the number of hidden layers to two. Figure 4.7 depicts the final structure setup for this model.

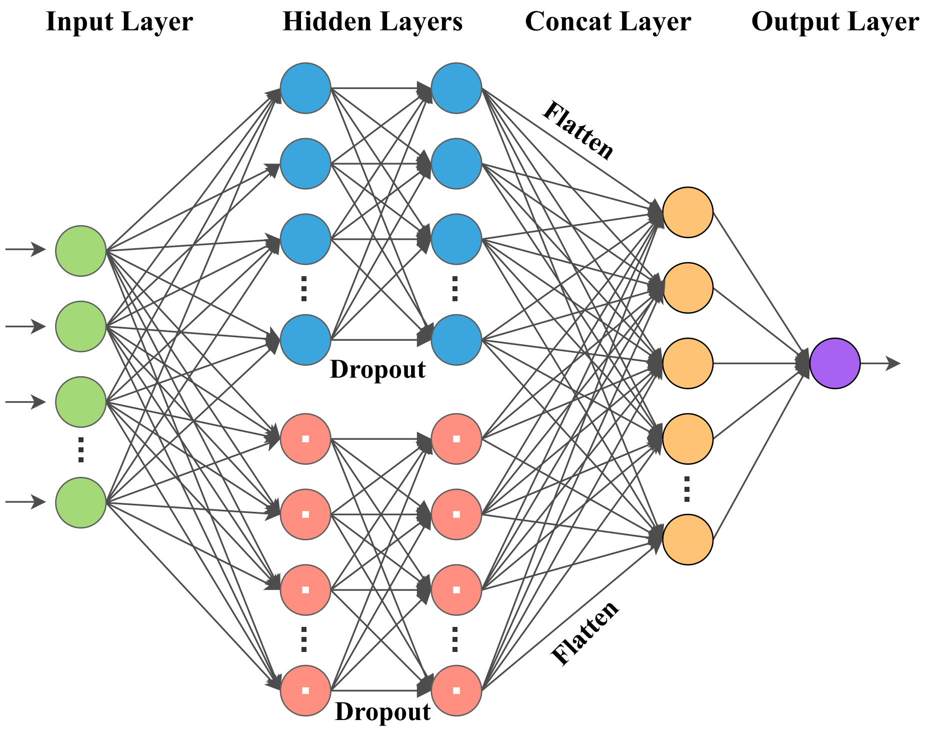

Figure 4.7: Graph for MLP&LSTM.

The training process of this model works of the same way that the previous models; however, the number of hidden layers is reduced to two and one concatenation layer is added in order to concatenate the outputs of the hidden layers of both DNNs. Then, the concatenated data pass thought the regularization dropout and output layers to obtain the final output that corresponds to the export prediction one month ahead after the input data.

4.4.2 LSTM&CNN

LSTM and CNN are combined to assemble this fifth neural network. Its structure and hiperparameters are the same as the described on Table 4.2 and Table 4.3 for LSTM and CNN, respectively; however, the number of hidden layers is two for LSTM, and for CNN there is only one convolutional and max-polling layer. Figure 4.8 depicts the final structure and parameters setup for this model.

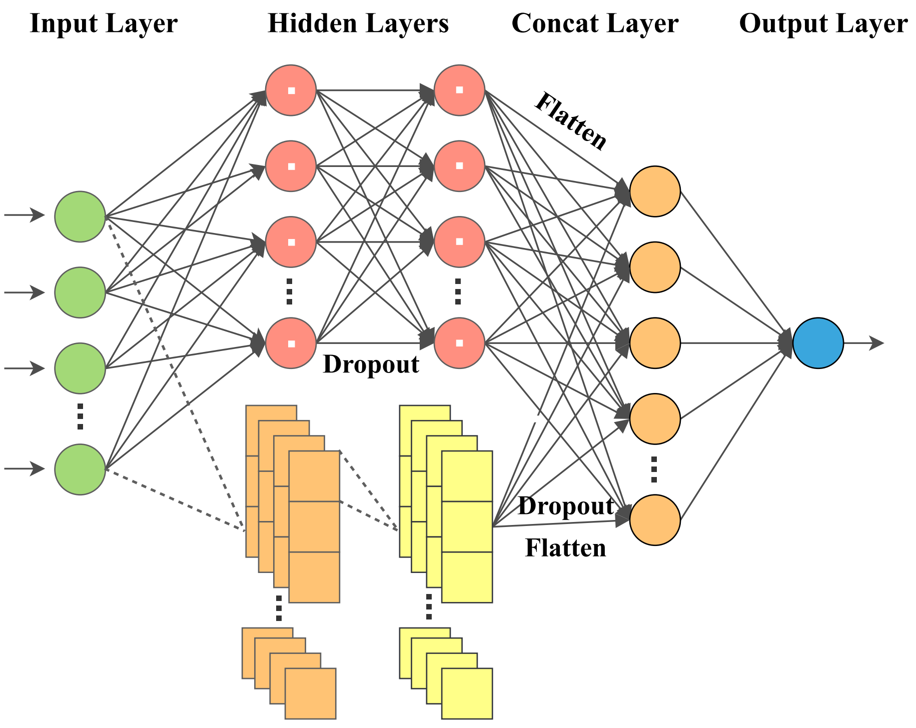

Figure 4.8: Graph for LSTM&CNN.

The training process and structure of this model works in the same way that the previous model described for MLP&LSTM, but instead of having a MLP there is a CNN assembled with one 1D convolutional and max pooling layer.

4.4.3 CNN&MLP

CNN and MLP are combined to assemble this last neural network. Its structure and parameters are the same as the described on Table 4.3 and Table 4.1 for CNN and MLP, respectively; however, the number of hidden layers is two for MLP, and for CNN there is only one convolutional and max-polling layer. Figure 4.9 depicts the final structure and parameters setup for this model.

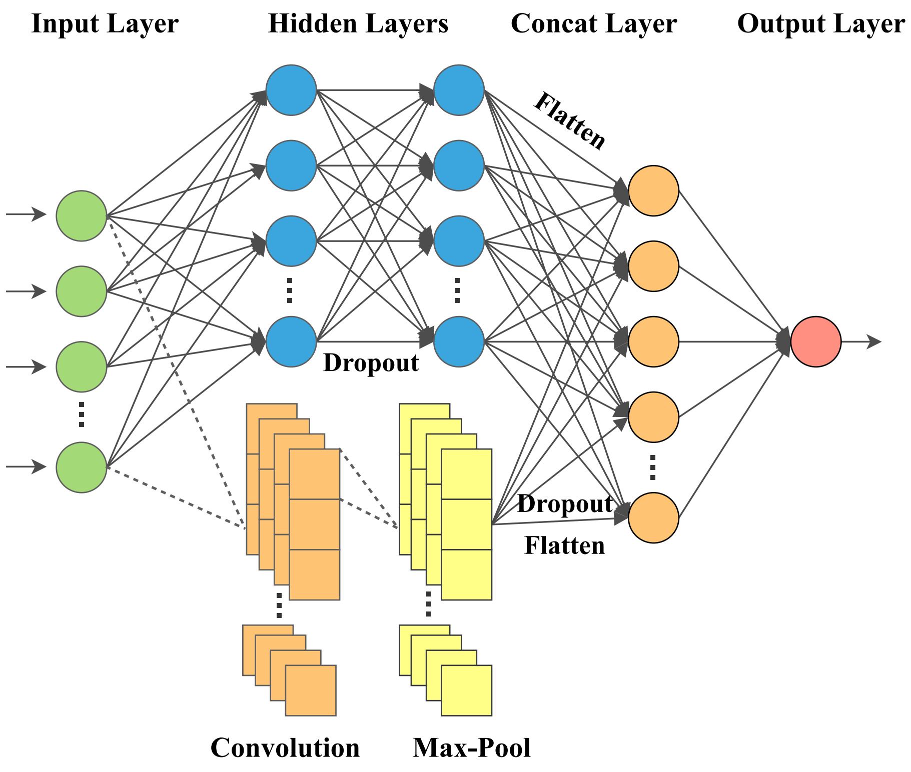

Figure 4.9: Graph for CNN&MLP.

The training process and structure of this model works in the same way that the previous model described for MLP&LSTM, but instead of having a LSTM there is a CNN assembled with one 1D convolutional and max pooling layer.

4.5 Metrics

In this section is explained the metrics selected to measure the performance and accuracy of the six DNNs proposed to predict the monthly banana exports. Then, the metrics used are: accuracy, window size and number of parameters.

4.5.1 Accuracy

Several metrics, such as is root mean squared error (RMSE), are used to measure the performance and accuracy of deep neural networks. RMSE is the standard deviation of the residuals; in other words, it is a measure of how spread out the residuals or prediction errors are around the line of best fit, [50]. As RMSE is mainly used for forecasting research, it has been considered such an excellent general-purpose error metric for numerical predictions, [50], for this project. RMSE is formulated as follows:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\widehat{y}_i - y_i)^2} \tag{4.2}$$

Where:

- $\widehat{y}_i$ is the predicted value
- y_i is the observation
- n is the number of observations available for analysis

4.5.2 Window size and number of parameters

Here it is considered a window size of 6, 12 and 24 values, since in subsection 4.2.2 it is mentioned that the size of the window and segment can increase until the less error approximation is reached, [20]. Moreover, when the window size increase, the number of parameters in the input layer increase as well; therefore, it means that there will be a higher number of parameters to train. In fact, training these parameters have a computational cost for the network, because it involves to compute a lot of floating point operations which are in charge of updating the weights and biases (parameters) of a DNN.

4.6 Overall system model

Once it was defined all the hyperparameters and configuration that it will be used, Figure 4.10 shows a flowchart of the overall system model proposed in this research. In general terms: first, we input the time series; second, we normalize the data and reframe it with three different window sizes; fourth, we feed the DNNs models with the preprocessed data; finally, we made the forecasting task and compare the results based on the established metrics.

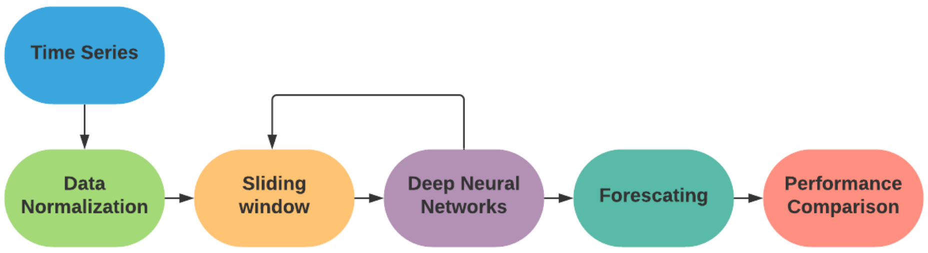

Figure 4.10: Flowchart of the overall system model

Chapter 5

Results and discussion

This chapter describes the results obtained after the application of the methodology with all the deep neural networks proposed. Moreover, in order to compare the behavior and accuracy of this networks, two different time series are analyzed.

5.1 Time series 1

The first time series studied is composed of the monthly banana exports, measured in metric tons, from January 2000 to August 2020 (see Figure 5.1). It is clear that this series has a growing tendency with seasonal and cyclical variations or movements.

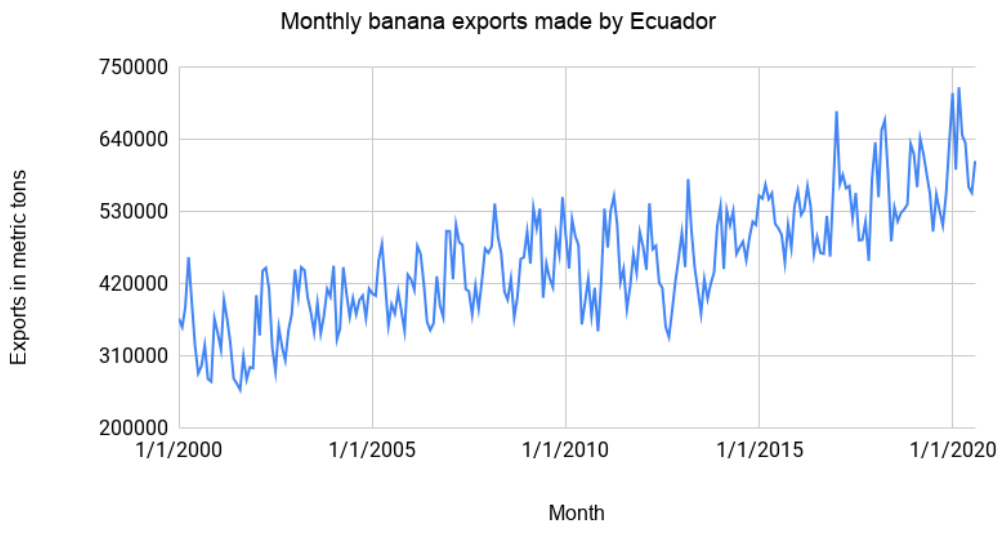

Figure 5.1: Time series 1

The results obtained with the six DNNs for this first time series are shown below:

5.1.1 Window size

Window size: 6 Epochs: 700							
Model/Error	**Number of Parameters**	**Loos Train**	**Loos Test**	**RMSE Train**	**RMSE Test**	**RMSE Train**	**RMSE Test**
MLP	169	0.010	0.040	0.083	0.169	46995.17	93161.06
LSTM	1.135	0.011	0.053	0.087	0.20	49582.42	107037.33
CNN	727	0.009	0.043	0.079	0.176	45167.48	96215.48
MLP & LSTM	827	0.009	0.037	0.081	0.156	45601.62	89383.41
CNN & LSTM	759	0.008	0.039	0.076	0.165	43317.58	91919.80
CNN & MLP	157	0.009	0.030	0.079	0.138	45804.62	80703.85

Table 5.1: Loss and RMSE for each model with a window size = 6

Table 5.1 shows that for the sequential or single models, CNN and MLP has the better results for each metric. It also shows that for the non-sequential or combined models, CNN & MLP is the best network.

Window size: 12 Epochs: 700							
Model/Error	**Number of Parameters**	**Loos Train**	**Loos Test**	**RMSE Train**	**RMSE Test**	**RMSE Train**	**RMSE Test**
MLP	211	0.008	0.017	0.077	0.108	43212.47	61832.31
LSTM	1.177	0.006	0.070	0.062	0.226	35976.25	122154.53
CNN	839	0.005	0.019	0.062	0.108	35744.58	63878.42
MLP & LSTM	911	0.006	0.010	0.064	0.076	36762.80	46699.41
CNN & LSTM	813	0.008	0.029	0.072	0.139	41507.42	79580.94
CNN & MLP	211	0.004	0.008	0.056	0.071	32584.83	43001.80

Table 5.2: Loss and RMSE for each model with a window size = 12

Table 5.2 shows that for the sequential or single models, CNN and MLP has the better results for each metric. It also shows that for the non-sequential or combined models, CNN & MLP is the best network.

Window size: 24 Epochs: 700							
Model/Error	**Number of Parameters**	**Loos Train**	**Loos Test**	**RMSE Train**	**RMSE Test**	**RMSE Train**	**RMSE Test**
MLP	295	0.004	0.011	0.054	0.079	31256.45	49731.17
LSTM	1.261	0.005	0.022	0.056	0.118	33028.82	69279.90
CNN	1.175	0.002	0.022	0.043	0.128	24995.99	68911.04
MLP & LSTM	1.079	0.004	0.026	0.053	0.129	30134.83	74645.47
CNN & LSTM	921	0.003	0.015	0.045	0.096	26558.19	57177.65
CNN & MLP	319	0.004	0.006	0.051	0.065	29956.30	37139.57

Table 5.3: Loss and RMSE for each model with a window size = 24

Table 5.2 shows that for the sequential or single models, CNN and MLP has the better results for each metric. It also shows that for the non-sequential or combined models, CNN & MLP is the best network.

In general, Tables 5.1, 5.2 and 5.3 show that for the sequential or single models, CNN and MLP present the best values for each metric; however, MLP overcomes to CNN with test results and CNN overcomes to MLP with the train results. It also shows that for the non-sequential or combined models, CNN & MLP has the better values. In fact, CNN & MLP is the best model in the three tables because its accuracy with the training and testing dataset are better than the rest of non-sequential and sequential models.

Moreover, Table 5.1, 5.2 and 5.3 show that while the window size increase, the Loss and RMSE for training and test dataset in all the models decrease. It also shows that CNN have a value of 0.022 and 0.128 for Loss test and RMSE test, respectively; however, Table5.2 it has values of 0.019 and 0.108. It means that, the Loss and RMSE value for the test dataset on CNN start to increase. The same pattern is seem for the CNN&MLP and MLP&LSTM.

Lastly, Table 5.1, 5.2 and 5.3 also show that while the window size increase, the number of parameters to train in each one of the models increase. Furthermore, when a model is combined with another one, the number of parameters can also increase or decrease. For instance, Table 5.3 shows that LSTM has 1.261 parameters to train; however, when it is combined with CNN, the number parameters to train decrease to 921 for CNN&LSTM. Another example is with MLP, which has 295 parameters and when it is combined with CNN, the number of parameters increase to 329 for CNN&MLP

5.1.2 Loss and RMSE

The results presented in the following subsections 5.1.2, 5.1.3, 5.1.4 and 5.1.5 are based on the configuration of sliding window size equal to 24.

Figure 5.2 depicts how the Loss value decreases for all the models during the train. It shows that CNN and CNN&MLP have the lower values (0.002 and 0.004 respectively, according to Table 5.3) and converge from around 200 epochs. It is also interesting see that MLP converges faster than the rest of models; however, from around 600 epochs, it start to increase its Loss values. In addition, Figure 5.2 shows that LSTM model needs from around 400 epochs to converge and has the highest Loss value, as it is also seem on Table 5.3 where its result is 0.005.

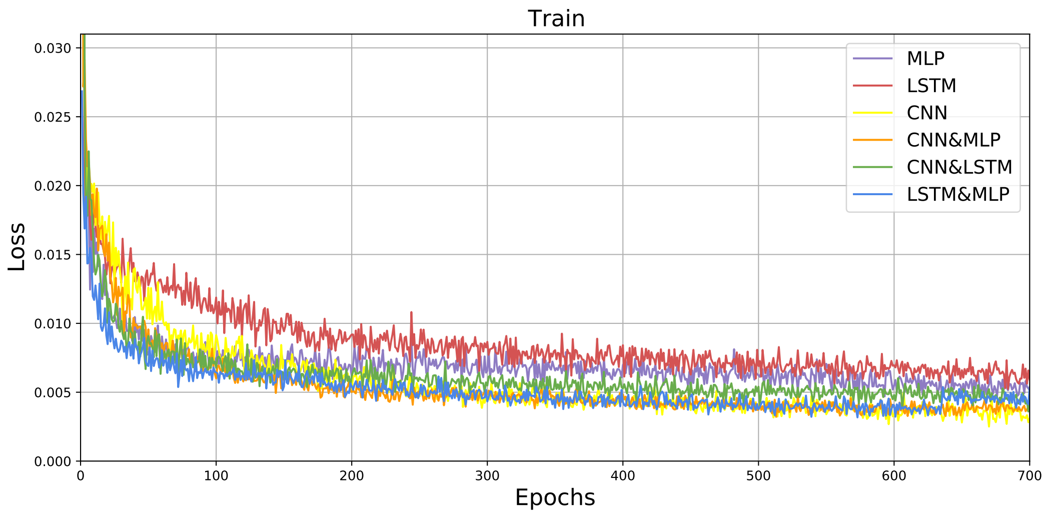

Figure 5.2: Loss function values for the six DNNs models

Figure 5.3 shows how the RMSE value decreases for all the models during the train. It shows that, for all the models, the RMSE values have a very similar behavior as on Figure 5.2; however, its results are highest. Then, CNN and and CNN&MLP have the lower values (0.043 and 0.051 respectively) and LSTM has the highest value (0.056). These RMSE values can be also seen on Table 5.3

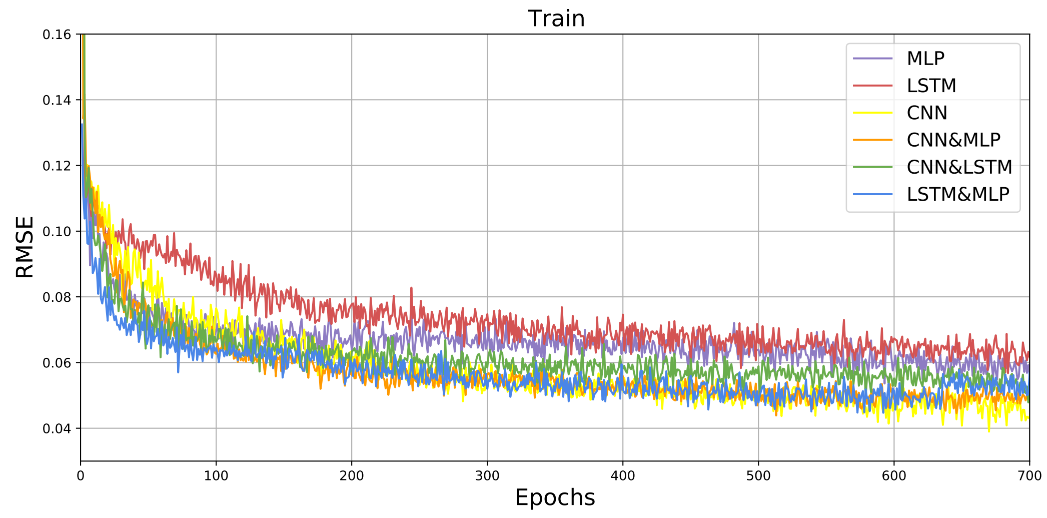

Figure 5.3: RMSE values for the six DNNs models

5.1.3 LSTM&MLP vs LSTM

Figure 5.4 shows the predicted values for the training dataset with LSTM and LSTM&MLP. LSTM seems have some smooth across the series, probably it needs more epochs of training since it has more parameter for training than MLP and LSTM&MLP. Nevertheless, it is appreciated that when LSTM is combined with MLP the smooth disappear and the results improve. Table 5.3 shows that the Loss and RMSE value for LSTM are 0.005 and 0.056 respectively, and for LSTM&MLP are 0.004 and 0.053. Moreover the difference between the predicted values and the real ones of LSTM and LSTM-MLP is 33k and 30k, respectively. Then, LSTM&MLP has better results on training.

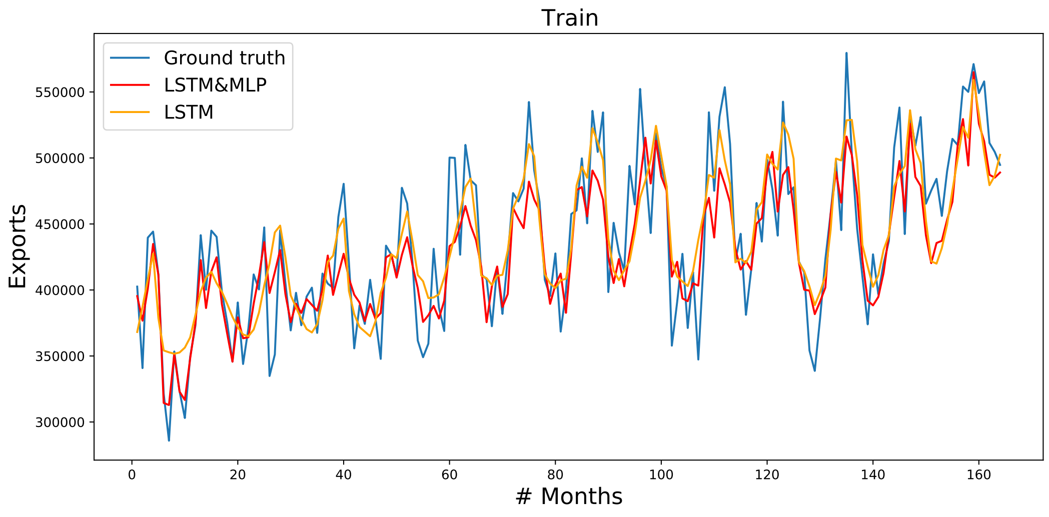

Figure 5.4: LSTM&MLP vs LSTM

Figure 5.5 shows the predicted values for the test dataset with LSTM and LSTM&MLP. From the 20 month the LSTM values start to drift away while the LSTM&MLP values are closer to the real ones. Table 5.3 shows that the Loss and RMSE value for LSTM are 0.022 and 0.118 respectively, and for MLP&LSTM are 0.026 and 0.129. Moreover the difference between the predicted values and the real ones of LSTM and LSTM&MLP is 69k and 74k, respectively. Then, LSTM has better results on test.

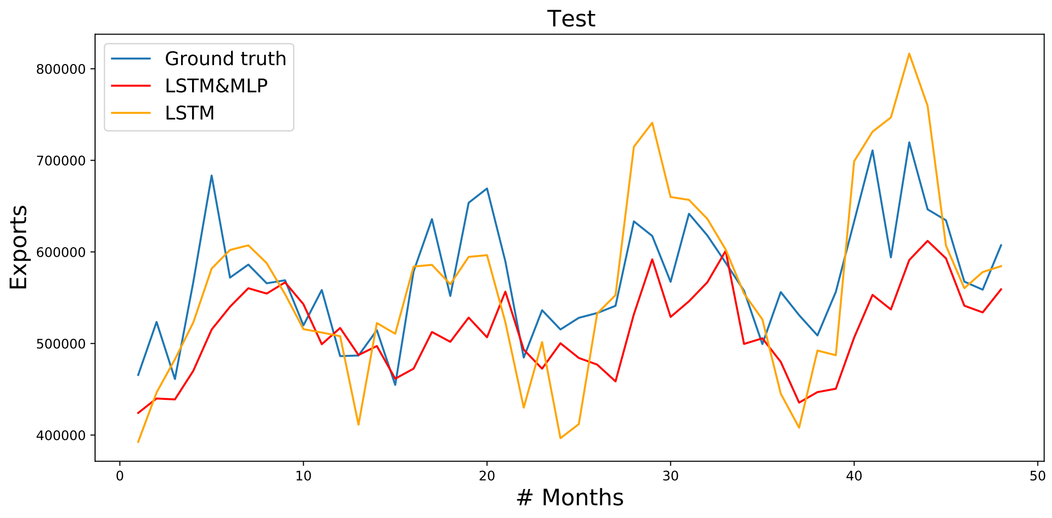

Figure 5.5: LSTM&MLP vs LSTM

5.1.4 CNN&LSTM vs CNN

Figure 5.6 shows the predicted values for the training dataset with CNN and CNN&LSTM. Both of the models shows a good performance and there are some regions where its values are overlapped with the real values. Table 5.3 shows that the Loss and RMSE value for CNN are 0.002 and 0.043 respectively, and for CNN&LSTM are 0.003 and 0.045. Moreover the difference between the predicted values and the real ones of CNN and CNN&LSTM is 24k and 26k, respectively. Then, CNN has better results on training.

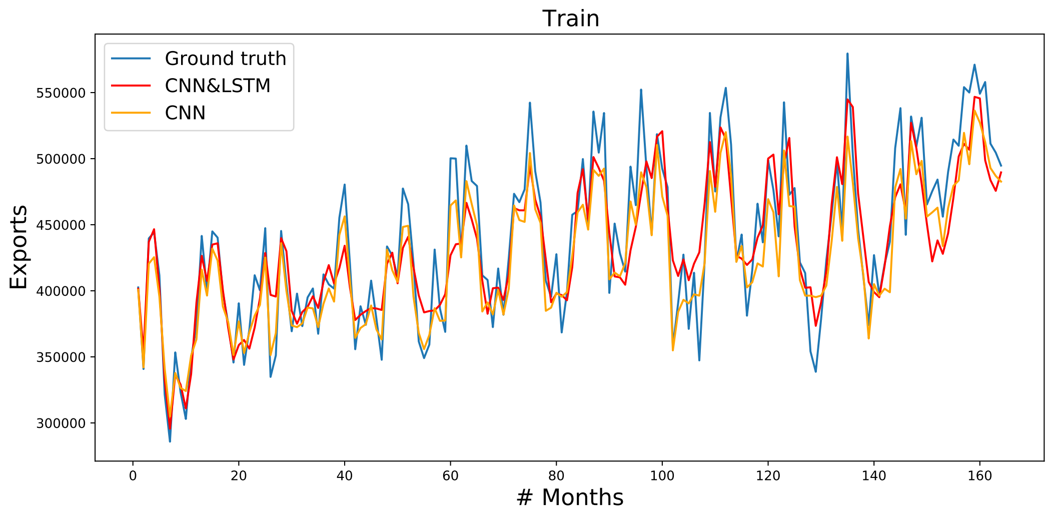

Figure 5.6: CNN&LSTM vs CNN

Figure 5.7 shows the predicted values for the test dataset with CNN and CNN&LSTM. The CNN&LSTM values are closer than CNN. Table 5.3 shows that the Loss and RMSE values for CNN are 0.022 and 0.128 respectively, and for CNN&LSTM are 0.015 and 0.096. Moreover the difference between the predicted values and the real ones of CNN and CNN-LSTM is 68k and 57k, respectively. Then, CNN&LSTM has better results on test.

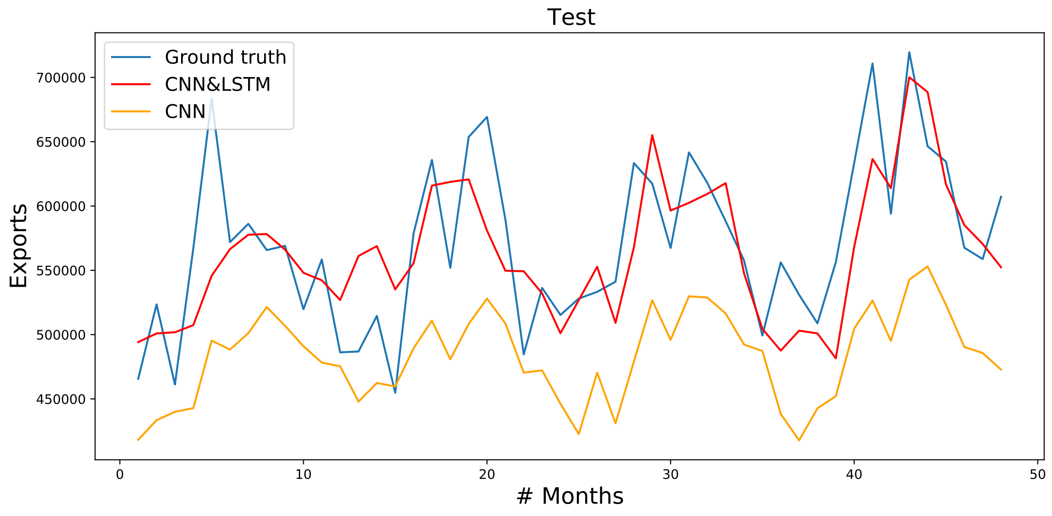

Figure 5.7: CNN&LSTM vs CNN

5.1.5 CNN&MLP vs MLP

Figure 5.8 shows the predicted values for the training dataset with MLP and CNN&MLP. Both of the models shows a good performance. It seems that combine CNN with MLP improve the results of MLP at the beginning of the series, then from 50 month its values seems very similar. Table 5.3 shows that the Loss and RMSE value for MLP are 0.004 and 0.054 respectively, and for CNN&MLP are 0.004 and 0.051. Moreover the difference between the predicted values and the real ones of MLP and CNN&MLP is 31k and 29k, respectively. Then, CNN&MLP has better results on training.

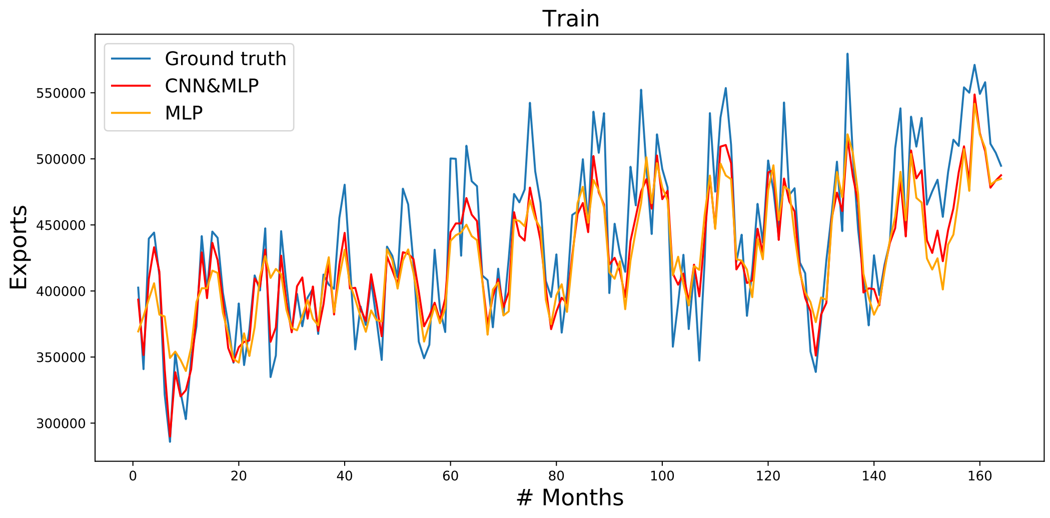

Figure 5.8: CNN&MLP vs MLP

Figure 5.9 shows the predicted values for the test dataset with MLP and CNN&MLP. Both of the models shows a good performance where its values are almost overlapped and close to the real ones. Table 5.3 shows that the Loss and RMSE value for MLP are 0.011 and 0.079 respectively, and for CNN&MLP are 0.006 and 0.065. Moreover the difference between the predicted values and the real ones of MLP and CNN&MLP is 49k and 37k, respectively. Then, CNN&MLP has better results on test.

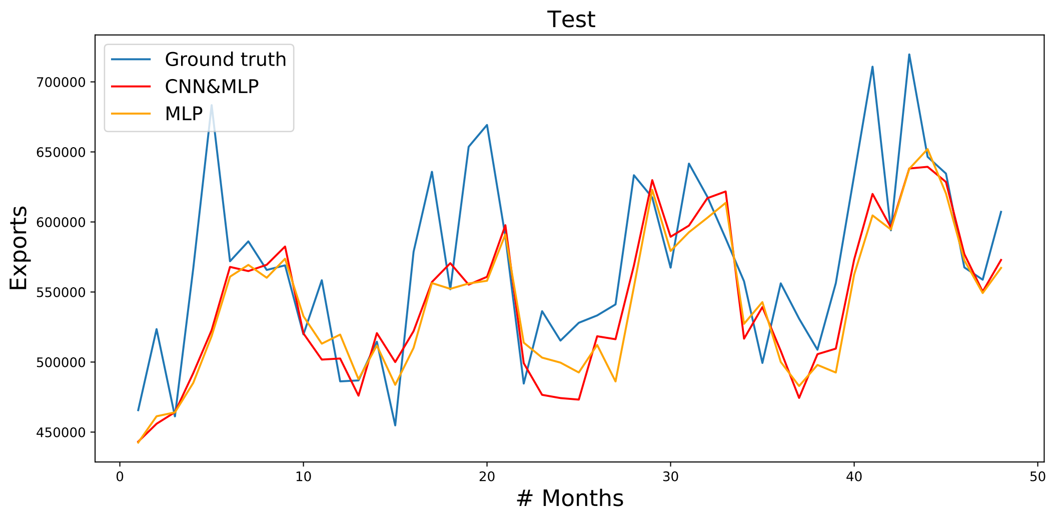

Figure 5.9: CNN&MLP vs MLP

5.2 Time series 2

The second time series studied is composed of the monthly banana exports, measured in thousands of USD FOB, from January 2000 to August 2020, see Figure 5.10. It is clear that this series has a growing tendency with seasonal and cyclical variations or movements.

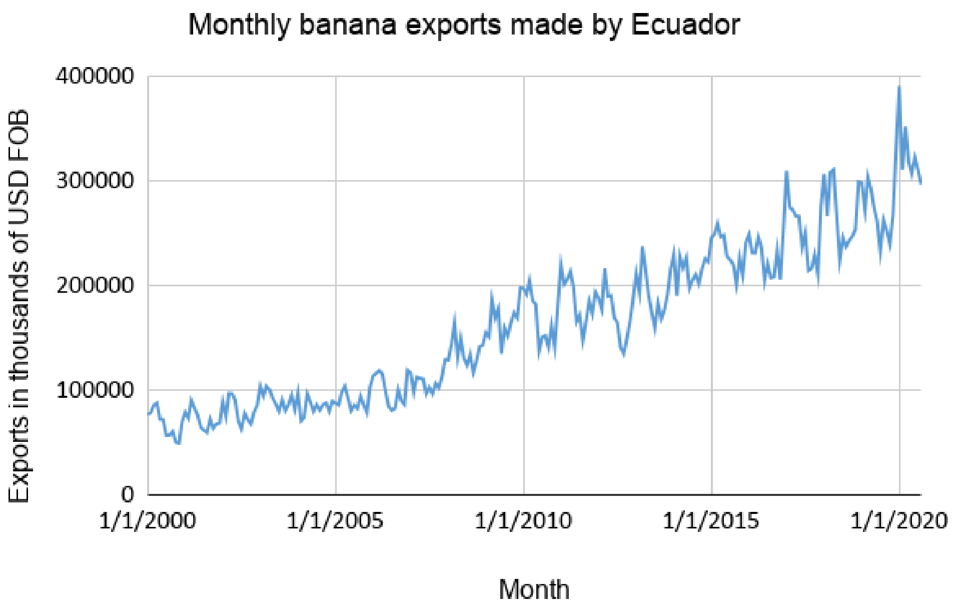

Figure 5.10: Time series 2

The results obtained with the six DNNs for this sencond time series are shown below:

5.2.1 Window size

Window size: 6
Epochs: 700

Model/Error	Number of Parameters	Loos Train	Loos Test	RMSE Train	RMSE Test	RMSE Train	RMSE Test
MLP	169	0.002	0.016	0.040	0.091	20987.32	49585.02
LSTM	1.135	0.004	0.037	0.048	0.170	24422.88	74031.50
CNN	727	0.002	0.027	0.039	0.144	20613.59	64145.16
MLP & LSTM	827	0.002	0.021	0.038	0.112	19581.64	55960.84
CNN & LSTM	759	0.002	0.014	0.036	0.085	18939.34	45964.15
CNN & MLP	157	0.002	0.013	0.037	0.077	19394.01	43873.97

Table 5.4: Loss and RMSE for each model with a window size = 6

Table 5.4 shows that for the sequential or single models, MLP has the better results for each metric. It also shows that for the non-sequential or combined models, CNN & MLP is the best network.

Window size: 12 **Epochs: 700**							
Model/Error	**Number of Parameters**	**Loos Train**	**Loos Test**	**RMSE Train**	**RMSE Test**	**RMSE Train**	**RMSE Test**
MLP	211	0.004	0.022	0.050	0.127	24773.12	57991.13
LSTM	1.177	0.002	0.010	0.033	0.065	17469.14	38923.86
CNN	839	0.002	0.014	0.040	0.089	20411.20	45836.32
MLP & LSTM	911	0.002	0.011	0.033	0.070	17351.91	41208.48
CNN & LSTM	813	0.001	0.012	0.029	0.078	15942.77	42111.40
CNN & MLP	211	0.001	0.009	0.028	0.063	15200.60	38370.19

Table 5.5: Loss and RMSE for each model with a window size = 12

Table 5.5 shows that for the sequential or single models, LSTM has the better results for each metric. It also shows that for the non-sequential or combined models, CNN & MLP is the best network.

Window size: 24 **Epochs: 700**							
Model/Error	**Number of Parameters**	**Loos Train**	**Loos Test**	**RMSE Train**	**RMSE Test**	**RMSE Train**	**RMSE Test**
MLP	295	0.001	0.010	0.030	0.064	15905.56	38768.74
LSTM	1.261	0.002	0.010	0.036	0.072	18785.04	40211.68
CNN	1.175	0.002	0.016	0.040	0.98	20787.56	48797.16
MLP & LSTM	1.079	0.001	0.009	0.031	0.060	16400.57	37723.77
CNN & LSTM	921	0.001	0.010	0.031	0.066	16080.10	38724.62
CNN & MLP	319	0.002	0.010	0.035	0.065	17809.02	39268.57

Table 5.6: Loss and RMSE for each model with a window size = 24

Table 5.6 shows that for the sequential or single models, MLP has the better results for each metric. It also shows that for the non-sequential or combined models, MLP & LSTM is the best network.

On the one hand, Tables 5.5 and 5.6 show that for the sequential or single models, LSTM and MLP present the best values for each metric. On the other hand, on Table 5.5 CNN overcomes to LSTM, since it starts with higher values than the rest of networks. Moreover, MLP is the only non-sequential model that improve its results when it is incremented the window size to 24 . For example, on Table 5.5 CNN has a value of 0.014 and 0.089 for Loss test and RMSE test, respectively; however, on Table 5.6 these values increase to 0.016 and 0.098. The same pattern occurs for results of LSTM.

Tables 5.4, 5.5, and 5.6 also shows that for the non-sequential or combined models, CNN&MLP and MLP&LSTM are the best models because its results for the training and testing dataset are better than the rest of non-sequential and sequential models. However, CNN & MLP is the best model when it has a window size of 12 and MLP & LSTM when it has a window size of 24. It means that, the results of CNN & MLP do not improve when the window size increase to 24.

On the one hand, Tables 5.4, 5.5, and 5.6 show that while the window size increase, the Loss and RMSE for training and test dataset in all the models can decrease or increase. On the other hand, these tables also show that while the size increase, the number of parameters to train of the models can increase or increase, just as it is decribed on section 5.1.1.

5.2.2 Loss and RMSE

The results presented in the following subsections 5.2.2, 5.2.3, 5.2.4 and 5.2.5 are based on the configuration of sliding window size equal to 24.

Figure 5.11 depicts how the Loss value decreases for all the models during the train. It shows that MLP, LSTM&MLP and CNN&MLP have the lowest values (0.001 according to Tables 5.6,5.6 and 5.2, respectively). Moreover, it is interesting see that all the models converge from around 100 epochs, unlike CNN which needs around 400 epochs. Moreover, the Loss train values for all the models vary from 0.001 and 0.002; however, if we consider the Loss test values, Table 5.6 shows that the results for CNN is 0.016 and for all the rest of models is 0.010.

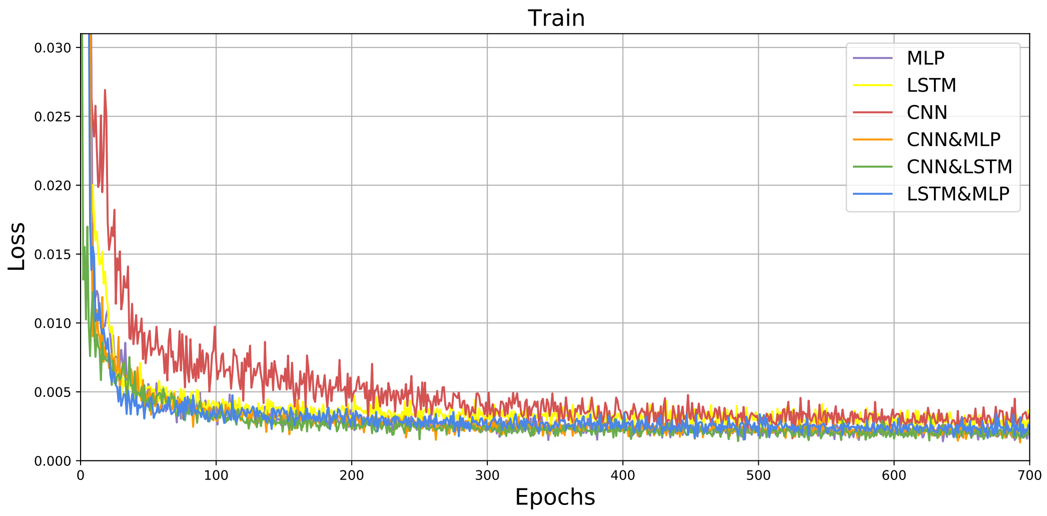

Figure 5.11: Loss function values for the six DNNs models

Figure 5.12 shows how the RMSE value decreases for all the models during the train. It shows that, for all the models, the RMSE values have a very similar behavior as on Figure 5.11; however, its values are higher. Then, MLP has has the lowest value (0.030) and CNN has the highest value (0.040). These RMSE values can be also seen on Table5.6

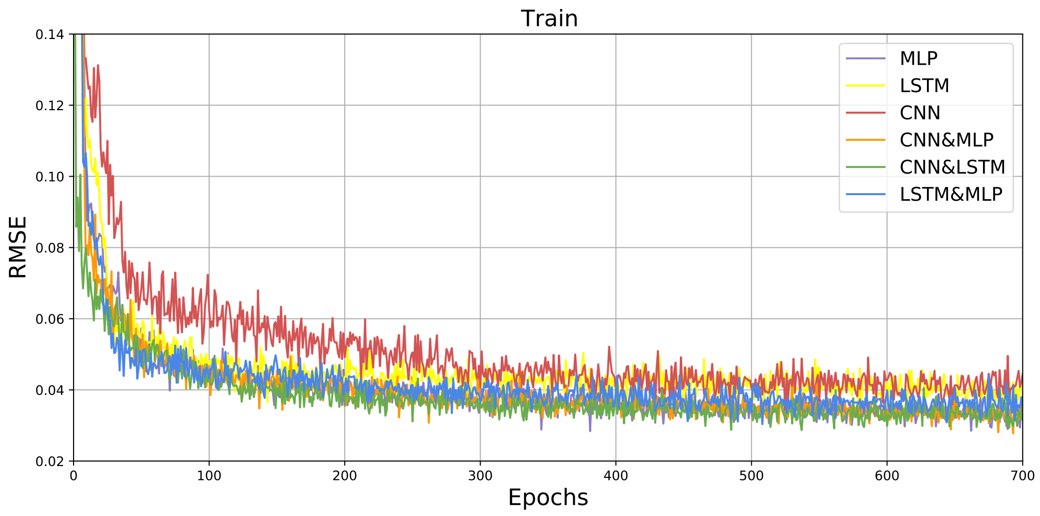

Figure 5.12: RMSE values for the six DNNs models

5.2.3 LSTM&MLP vs LSTM

Figure 5.13 shows the predicted values for the training dataset with LSTM and LSTM&MLP. Both of the models seems have some smooth until the month 60, then from that mounth the values seems closer to the real ones. Nevertheless, it is appreciated that when LSTM is combine with MLP some of the smooth disappear and the results improve. Table 5.6 shows that the Loss and RMSE values for LSTM are 0.002 and 0.036 respectively, and for LSTM&MLP are 0.001 and 0.031. Moreover the difference between the predicted values and the real ones of LSTM and LSTM-MLP is 15k and 16k, respectively. Then, LSTM&MLP has better results on training.

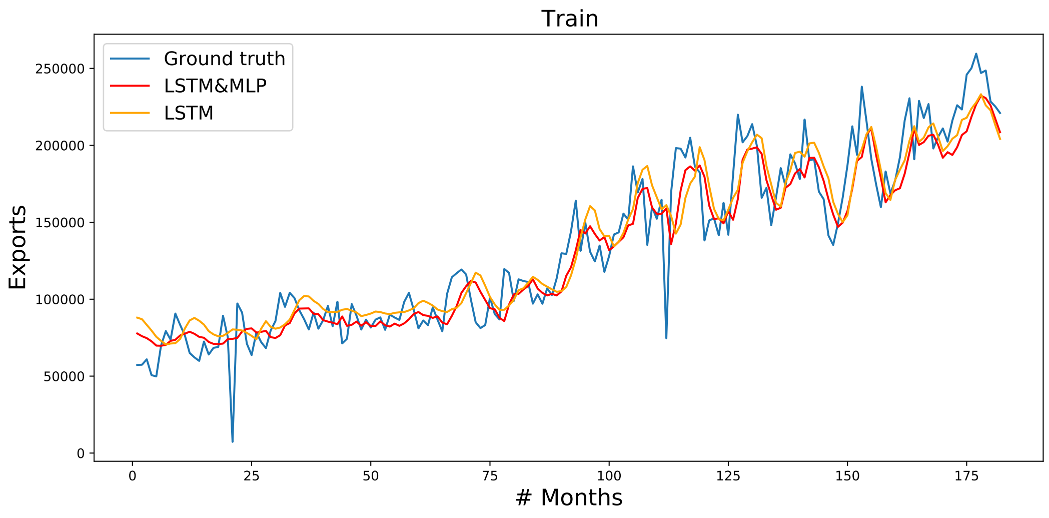

Figure 5.13: LSTM&MLP vs LSTM

Figure 5.14 shows the predicted values for the test dataset with LSTM and LSTM&MLP. The LSTM&MLP values are closer to the real ones, than LSTM. Table 5.6 shows that the Loss and RMSE values for LSTM are 0.010 and 0.072 respectively, and for MLP&LSTM are 0.009 and 0.060. Moreover the difference between the predicted values and the real ones of LSTM and MLP&LSTM is 40k and 37k, respectively. Then, MLP&LSTM has better results on test.

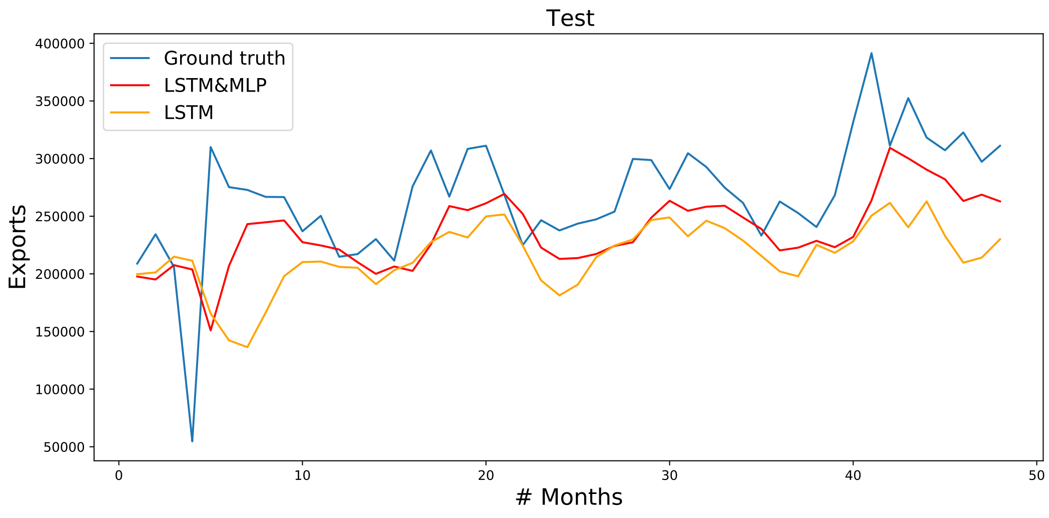

Figure 5.14: LSTM&MLP vs LSTM

5.2.4 CNN&LSTM vs CNN

Figure 5.15 shows the predicted values for the training dataset with CNN and CNN&LSTM. Both of the models shows a good performance. It seems that combine LSTM with CNN improve the results of CNN since at the beginning of the series CNN have a straight trend until the 60 month, then from that month the values of both models seems very similar. Table 5.6 shows that the Loss and RMSE value for CNN are 0.002 and 0.040 respectively, and for CNN&LSTM are 0.001 and 0.031. Moreover the difference between the predicted values and the real ones of CNN and CNN&LSTM is 20k and 16k, respectively. Then, CNN&LSTM has better results on training.

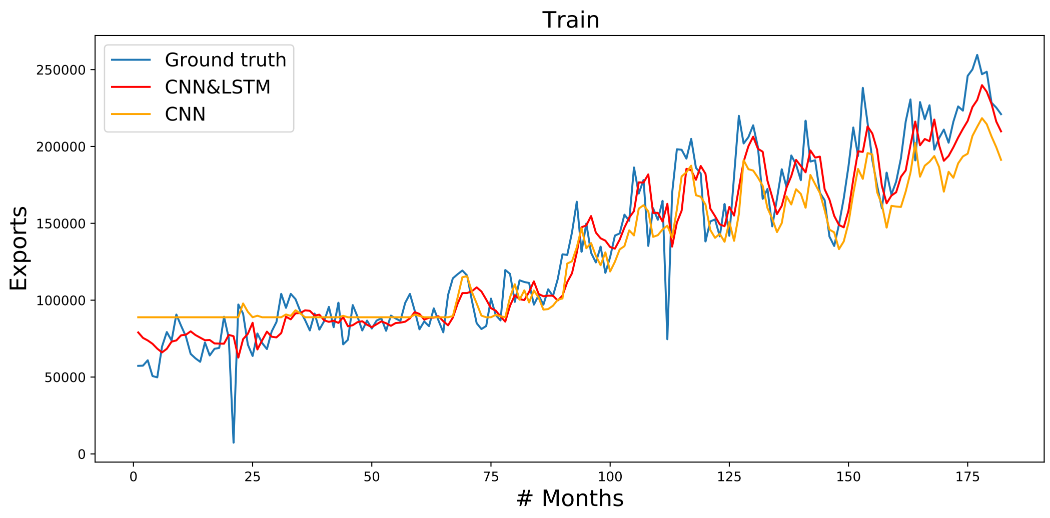

Figure 5.15: CNN&LSTM vs CNN

Figure 5.16 shows the predicted values for the test dataset with CNN and CNN&LSTM. CNN seems have, until the 10 month, better values; however, from that month the CNN&LSTM values are closer than CNN. Table 5.6 shows that the Loss and RMSE values for CNN are 0.016 and 0.098 respectively, and for CNN&LSTM are 0.010 and 0.066. Moreover the difference between the predicted values and the real ones of CNN and CNN-LSTM is 48k and 38k, respectively. Then, CNN&LSTM has better results on test.

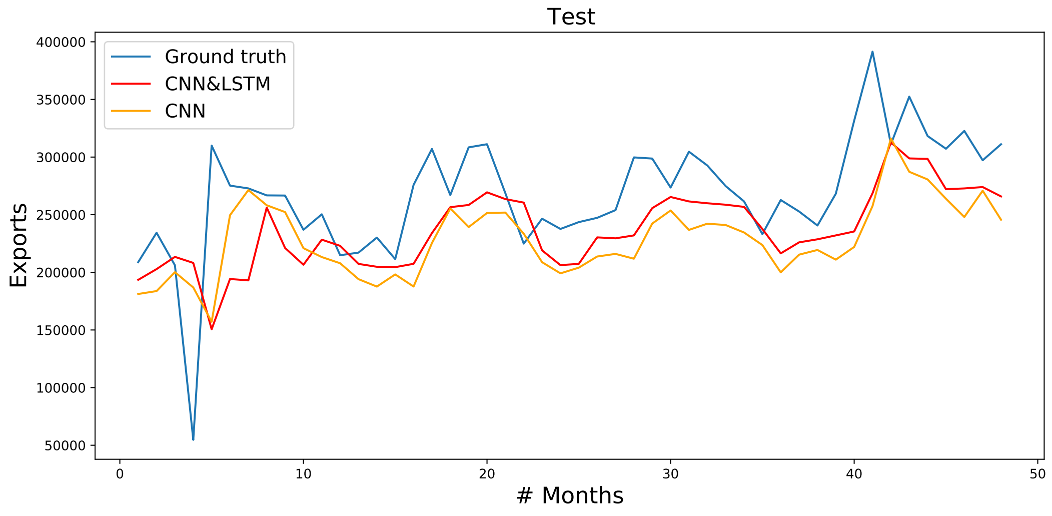

Figure 5.16: CNN&LSTM vs CNN

5.2.5 CNN&MLP vs MLP

Figure 5.17 shows the predicted values for the training dataset with MLP and CNN&MLP. Both of the models shows a good performance. It seems combine CNN with MLP improve the results of MLP across all the series. Table 5.6 shows that the Loss and RMSE values for MLP are 0.001 and 0.030 respectively, and on Table 5.5 for CNN&MLP are 0.001 and 0.028. Moreover the difference between the predicted values and the real ones of MLP and CNN&MLP is 15.9k and 15.2k, respectively. Then, CNN&MLP has better results on training. In this case are considered Tables 5.5 and 5.6 because MLP has better performance with a window size of 24 and CNN&MLP with a window size of 12.

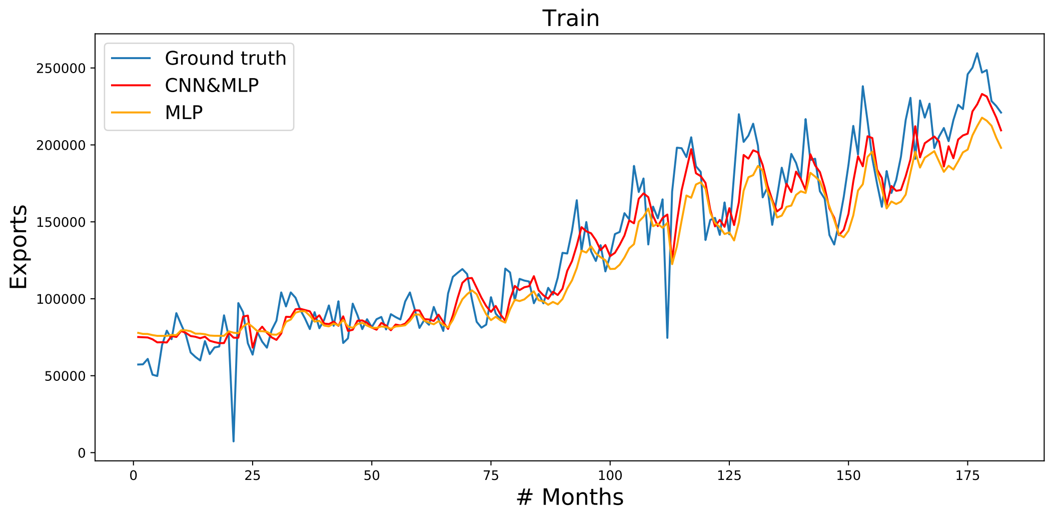

Figure 5.17: CNN&MLP vs MLP

Figure 5.18 shows the predicted values for the test dataset with MLP and CNN&MLP. Both of the models shows a good performance where its values are almost overlapped and close to the real ones. Table 5.6 shows that the Loss and RMSE value for MLP are 0.010 and 0.064 respectively, and on Table 5.5 for CNN&MLP are 0.009 and 0.063. Moreover the difference between the predicted values and the real ones of MLP and CNN&MLP is 38.7k and 38.3k, respectively. Then, CNN&MLP has better results on test. In this case are considered Tables 5.5 and 5.6 because MLP has better performance with a window size of 24 and CNN&MLP with a window size of 12.

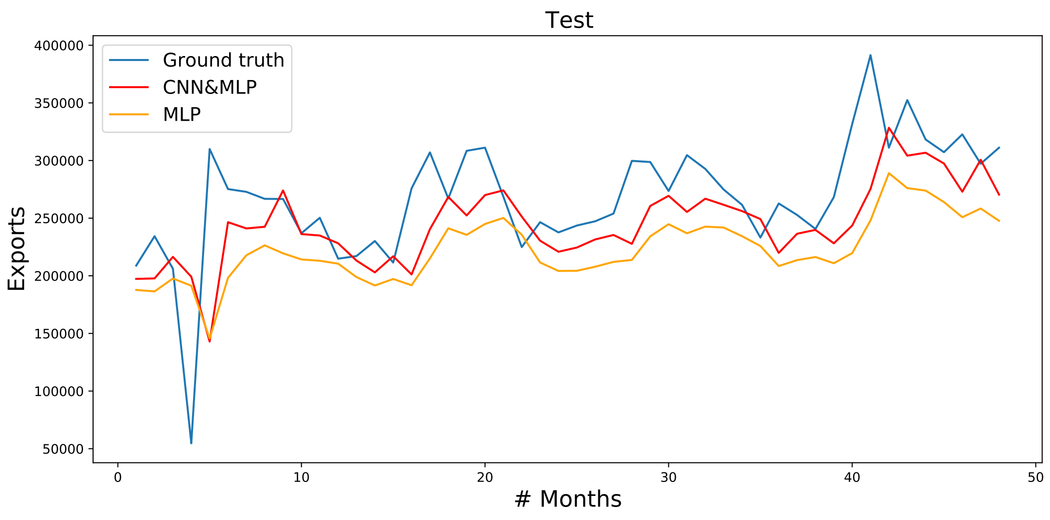

Figure 5.18: CNN&MLP vs MLP

Chapter 6

Conclusions

In general terms, the main goal of this project is to implement, evaluate and compare the performance and accuracy of the combination of three different DNNs structures, in order to do a time series prediction. At a more technical and specific level, the following conclusions are listed below.

First, it was proved that, increasing the sliding window size improved the performance of a DNN because it allowed to compute better results in metrics such as RMSE and Loss. On the one hand, an sliding window of size 12 is the optimal value calculated in order to do exports predictions with models such as CNN, CNN&MLP and MLP&LSTM. On the other hand, increasing this hyperparameter to 24 is better for models such as MLP, LSTM and CNN&LSTM.

Second, it was demonstrated that the proposed non-sequential DNNs overcame the performance of the sequential ones because they combined the capacity of inference of two or more different networks. For instance, the network that obtained the best results regarding RMSE and Loss(in the training and testing stage) is the non-sequential CNN&MLP, whose layers were assembled in a parallel structure from two different models, such as MLP and CNN, that also shown excellent values among the sequential networks.

Third, it was proved the computational complexity of proposed non-sequential DNNs is lesser compared with the sequential ones because they had a small number of trainable parameters. For instance, the best model presented in this work (MLP&CNN) had a total of 319 trainable parameters and shown a superior performance over other networks, such as CNN and LSTM, which had more than 1000 trainable parameters

Finally in this project,for time series prediction, a novel and effective method based on the combination of deep neural networks was developed and presented. This method produces good results and successfully predicts the monthly banana exports made by Ecuador. This AI tool is important for a country where banana export represent the second source of incomes for its developing economy.

Future work

As future work it is possible to analyze the performance and accuracy of these same networks or others, but this time instead of having two parallel networks, it may be include include the wide path. The wide path helps to the network to learn both simple rules (through the short path) and deep patterns (using the deep path); therefore, assemble a DNN including this structure can be beneficial, in order to know if it is possible to obtain better results that the ones presented in this work.

In this project are also presented the predictions of several DNNs with a time series of just one variable. However, as future work it is possible to scale the data processing and change the algorithms in order to do forecasting of a multivariate time series with the support of Tensorflow&Keras (or any other tool) that provides a good environment to train and evaluate DNNs.

The structure and hyperparameters plays a important role during the assembling, training and evaluating of a DNN. Therefore, as future work it is also recommended to vary the values of each one of these features in all the DNNs presented. In this way, it is possible to found a better configuration for all the DNNs and improve the accuracy and performance that these networks shown in this project.

Finally, the web page of Central Bank of Ecuador release new exports values every month; in fact, it is possible to found many information about the several products that Ecuador exports, in dollars or boxes. Therefore, the study and analyze of forecasting the Ecuadorian exports of any product, such as bananas in this case, is a rich source for further research.

Bibliography

[1] M. Imdadullah, "Basic statistics and data analysis," 2014.

[2] S. Agatonovic-Kustrin and R. Beresford, "Basic concepts of artificial neural network (ann) modeling and its application in pharmaceutical research," *Journal of pharmaceutical and biomedical analysis*, vol. 22, no. 5, pp. 717–727, 2000.

[3] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems.* O'Reilly Media, 2019.

[4] F. Liu and Y. Deng, "A fast algorithm for network forecasting time series," *IEEE Access*, vol. 7, pp. 102 554–102 560, 2019.

[5] S. Siami-Namini, N. Tavakoli, and A. S. Namin, "A comparison of arima and lstm in forecasting time series," in *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA).* IEEE, 2018, pp. 1394–1401.

[6] R. Sen, H.-F. Yu, and I. Dhillon, "Think globally, act locally: A deep neural network approach to high-dimensional time series forecasting," *arXiv preprint arXiv:1905.03806*, 2019.

[7] S. Huang, D. Wang, X. Wu, and A. Tang, "Dsanet: Dual self-attention network for multivariate time series forecasting," in *Proceedings of the 28th ACM international conference on information and knowledge management*, 2019, pp. 2129–2132.

[8] S. S. Rangapuram, M. Seeger, J. Gasthaus, L. Stella, Y. Wang, and T. Januschowski, "Deep state space models for time series forecasting," in *Proceedings of the 32nd international conference on neural information processing systems*, 2018, pp. 7796–7805.

[9] M. Sit and I. Demir, "Decentralized flood forecasting using deep neural networks," *arXiv preprint arXiv:1902.02308*, 2019.

[10] G. M. Mayón Mendoza, "Análisis de la variación de las exportaciones de banano ecuatoriano en el periodo 2014-2018." 2020.

[11] A. Priscila, "Inversiones reflejan en un mejor desempeño," *Revista Líderes*, 2020.

[12] F. Eckert, R. J. Hyndman, and A. Panagiotelis, "Forecasting swiss exports using bayesian forecast reconciliation," *European Journal of Operational Research*, vol. 291, no. 2, pp. 693–710, 2021.

[13] R. Adhikari and R. K. Agrawal, "An introductory study on time series modeling and forecasting," *arXiv preprint arXiv:1302.6613*, 2013.

[14] A. K. Jain, J. Mao, and K. M. Mohiuddin, "Artificial neural networks: A tutorial," *Computer*, vol. 29, no. 3, pp. 31–44, 1996.

[15] S. B. Maind, P. Wankar *et al.*, "Research paper on basic of artificial neural network," *International Journal on Recent and Innovation Trends in Computing and Communication*, vol. 2, no. 1, pp. 96–100, 2014.

[16] B. D. Ripley, *Pattern recognition and neural networks.* Cambridge University press, 2007.

[17] S. Sharma, "Activation functions in neural networks," *Towards Data Science*, vol. 6, 2017.

[18] E. Y. Li, "Artificial neural networks and their business applications," *Information & Management*, vol. 27, no. 5, pp. 303–313, 1994.

[19] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal processing magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[20] H. Hota, R. Handa, and A. Shrivas, "Time series data prediction using sliding window based rbf neural network," *International Journal of Computational Intelligence Research*, vol. 13, no. 5, pp. 1145–1156, 2017.

[21] M. W. Gardner and S. Dorling, "Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences," *Atmospheric environment*, vol. 32, no. 14-15, pp. 2627–2636, 1998.

[22] E. Eğrioğlu, Ç. H. Aladağ, and S. Günay, "A new model selection strategy in artificial neural networks," *Applied Mathematics and Computation*, vol. 195, no. 2, pp. 591–597, 2008.

[23] H. Ramchoun, M. A. J. Idrissi, Y. Ghanou, and M. Ettaouil, "Multilayer perceptron: Architecture optimization and training." *IJIMAI*, vol. 4, no. 1, pp. 26–30, 2016.

[24] D. Brezak, T. Bacek, D. Majetic, J. Kasac, and B. Novakovic, "A comparison of feed-forward and recurrent neural networks in time series forecasting," in *2012 IEEE Conference on Computational Intelligence for Financial Engineering & Economics (CIFEr).* IEEE, 2012, pp. 1–6.

[25] X. Ma, Z. Tao, Y. Wang, H. Yu, and Y. Wang, "Long short-term memory neural network for traffic speed prediction using remote microwave sensor data," *Transportation Research Part C: Emerging Technologies*, vol. 54, pp. 187–197, 2015.

[26] O. A. Abidogun, "Data mining, fraud detection and mobile telecommunications: call pattern analysis with unsupervised neural networks," Ph.D. dissertation, University of the Western Cape, 2005.

[27] J. Riofrıo-Valarezo, D. H. Peluffo-Ordónez, and O. Chang, “Forecasting consumer price index (cpi) of ecuador: A comparative study of predictive models,” *International Journal on Advanced Science Engineering Information Technology*, vol. 10, no. 3, pp. 1078–1084, 2020.

[28] K. O’Shea and R. Nash, “An introduction to convolutional neural networks,” *arXiv preprint arXiv:1511.08458*, 2015.

[29] Z. Wang, W. Yan, and T. Oates, “Time series classification from scratch with deep neural networks: A strong baseline,” in *2017 International joint conference on neural networks (IJCNN)*. IEEE, 2017, pp. 1578–1585.

[30] A. Borovykh, S. Bohte, and C. W. Oosterlee, “Conditional time series forecasting with convolutional neural networks,” *arXiv preprint arXiv:1703.04691*, 2017.

[31] J. Brownlee, “Machine learning mastery with python,” *Machine Learning Mastery Pty Ltd*, vol. 527, pp. 100–120, 2016.

[32] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.

[33] M. Abadi *et al.*, “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[34] H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir *et al.*, “Wide & deep learning for recommender systems,” in *Proceedings of the 1st workshop on deep learning for recommender systems*, 2016, pp. 7–10.

[35] J. A. Pérez *et al.*, “Estudio de factibilidad para el establecimiento de una exportadora de banano en guayaquil, ecuador, para su comercialización en alemania,” 2018.

[36] J. Horák, P. Šuleř, and J. Vrbka, “Comparison of neural networks and regression time series when predicting the export development from the usa to prc,” in *International Scientific Conference ”Contemporary Issues in Business, Management and Economics Engineering”*, 2019.

[37] O. Chang, G. Mosquera, Z. Castillo, and L. Zhinin-Vera, “Sales forecast by using deep rectifier network,” in *Proceedings of the Future Technologies Conference*. Springer, 2020, pp. 378–389.

[38] M. Hao, J. Fu, D. Jiang, F. Ding, and S. Chen, “Simulating the linkages between economy and armed conflict in india with a long short-term memory algorithm,” *Risk Analysis*, vol. 40, no. 6, pp. 1139–1150, 2020.

[39] C. Vidya and K. Prabheesh, “Implications of covid-19 pandemic on the global trade networks,” *Emerging Markets Finance and Trade*, vol. 56, no. 10, pp. 2408–2421, 2020.

[40] S. A. Haider, S. R. Naqvi, T. Akram, G. A. Umar, A. Shahzad, M. R. Sial, S. Khaliq, and M. Kamran, "Lstm neural network based forecasting model for wheat production in pakistan," *Agronomy*, vol. 9, no. 2, p. 72, 2019.

[41] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *2017 International Conference on Engineering and Technology (ICET)*. Ieee, 2017, pp. 1–6.

[42] K. H. Kouassi and D. Moodley, "An analysis of deep neural networks for predicting trends in time series data," in *Southern African Conference for Artificial Intelligence Research*. Springer, 2021, pp. 119–140.

[43] P. Li, K. Zhou, X. Lu, and S. Yang, "A hybrid deep learning model for short-term pv power forecasting," *Applied Energy*, vol. 259, p. 114216, 2020.

[44] X. Huang, Q. Li, Y. Tai, Z. Chen, J. Zhang, J. Shi, B. Gao, and W. Liu, "Hybrid deep neural model for hourly solar irradiance forecasting," *Renewable Energy*, vol. 171, pp. 1041–1060, 2021.

[45] H. Sohn and C. R. Farrar, "Damage diagnosis using time series analysis of vibration signals," *Smart materials and structures*, vol. 10, no. 3, p. 446, 2001.

[46] Q. Zhang and S. Sun, "Weighted data normalization based on eigenvalues for artificial neural network classification," in *International Conference on Neural Information Processing*. Springer, 2009, pp. 349–356.

[47] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[48] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[49] K. Zakka, "Cs231n convolutional neural networks for visual recognition," https://github.com/cs231n/cs231n.github.io/blob/master/convolutional-networks.md, 2020.

[50] S. P. Neill and M. R. Hashemi, *Fundamentals of ocean renewable energy: generating electricity from the sea*. Academic Press, 2018.