Forecasting time series by using deep learning techniques

[1] Oscar Vega
Yachay Tech
HPC Course
oscar.vega@yachaytech.edu.ec
Overleaf link https://es.overleaf.com/8318441177qjccnkrfwhrv

***Abstract*—This document presents the importance of forecasting and time series in the banana exports industry. Propose the sliding window method to convert a data vector into a data matrix and make forecasting. Compares RMSE values between six Deep neural Networks: three linear models of MLP, LSTM and CNN, and three non-linear models which combine MLP, LSTM and CNN in a parallel way of two columns. Finally, is considered Mirrored Strategy as one benchmark to make Data parallelism on big scale projects.**

***Index Terms*—Deep Neural Networks, Time series, Sliding window, Forecasting, LSTM, MLP, CNN, Data Parallelism, Mirrored Strategy.**

I. Introduction

Forecasting and time series analysis are relevant topics because they can help to explore and analyze different characteristics of data related to some fields such as traffic, finance, engineering, complex networks, and so on [1]. 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 [2]. For example, predicting project costs can help individuals and organizations reduce costs and schedule projects.

Deep neural networks (DNN) have received an increasing amount of attention in time series analysis. Several studies where DNN has been used for time-series data have show great results due to its the capability to extract higher order features,identify complex patterns within and across time series, and can do so from data sets of raw time series with considerably less human effort [3]. Moreover, this studies provides an extensive vision in increasing the usage of neural networks architectures in time series data [4].

A. Univariate Forecasting

A time series is a set of observations measured sequentially through time. The term ”univariate” refers to a time series that consists of single observations recorded sequentially through time (every hour, day, week, month, year, or any other regular interval) without taking into account the effect of the other variables.

Identify applicable funding agency here. If none, delete this.

B. Sliding window method

Sliding Window is a temporary approximation over the actual value of the time series data. After selecting the first segment, the next segment is selected from the end of the first segment. The process is repeated until all time series data are segmented. For instance, on Fig.1 the sliding window size is equal to 12 and it accumulate the historical time series data used to predict next year of banana exports.

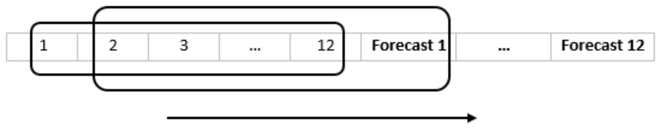


Fig. 1. Window size = 12.

C. Long short-term memory neural network

LSTM is a unique type of Recurrent Neural Network (RNN) capable of learning long-term dependencies, which is useful for certain types of prediction that require the network to retain information over longer time periods, a task that traditional RNNs struggle with. LSTM module has 3 gates named as Forget gate, Input gate and Output gate, as on Fig2

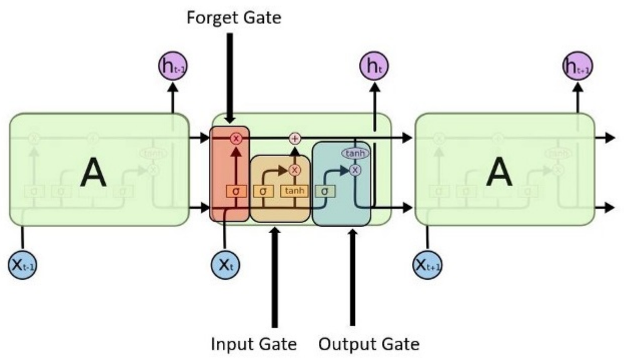


Fig. 2. Long short-term memory structure

- Forget Gate: Controls what information to throw away from memory and decides how much of the past it should remember.

- Update/Input Gate: Controls what new information is added to cell state from current input and decides how much of this unit is added to the current state.
- Output Gate: Conditionally decides what to output from the memory and in which part of the current cell makes it to the output.

D. Convolutional Neural Network

CNN is a deep neural network originally designed for image analysis. Recently, it was discovered that the CNN also has an excellent capacity in sequent data analysis such as natural language processing (Zhang, 2015). CNN always contains two basic operations, namely convolution and pooling. The convolution operation using multiple filters is able to extract features (feature map) from the data set, through which their corresponding spatial information can be preserved. The pooling operation, also called subsampling, is used to reduce the dimensionality of feature maps from the convolution operation. Max pooling and average pooling are the most common pooling operations used in the CNN. Due to the complicity of CNN, relu is the common choice for the activation function to transfer gradient in training by backpropagation Fig.3

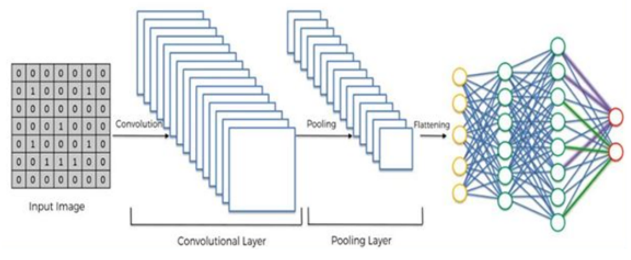


Fig. 3. Convolutional Neural Network structure

E. Multilayer Perceptron

Multi layer perceptron (MLP) is a supplement of feed forward neural network. It consists of three types of layers—the input layer, output layer and hidden layer Fig.4 The input layer receives the input signal to be processed. The required task such as prediction and classification is performed by the output layer. An arbitrary number of hidden layers that are placed in between the input and output layer are the true computational engine of the MLP.

Similar to a feed forward network in a MLP the data flows in the forward direction from input to output layer. The neurons in the MLP are trained with the back propagation learning algorithm. MLPs are designed to approximate any continuous function and can solve problems which are not linearly separable. The major use cases of MLP are pattern classification, recognition, prediction and approximation. [5]

II. Problem

On the last years, several DNN's like Multi-Layer Perceptron (MLP), Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM) model, between others, have been deployed for predicting time series data, and have caught the attention of many economic agents which are interesting in export forecast because it provides valuable information and can support economic policy makers, monetary authorities, and exporting firms in making decisions [5]. Therefore, in this projects it is develop four deep neural networks, LTM and MLP: both in linear and non-linear way, in order to know which offers best predictions with respect to lower forecast errors and higher accuracy of forecasts by applying a specific data set of the number of monthly exports made by Ecuador between the years 2012-2020.

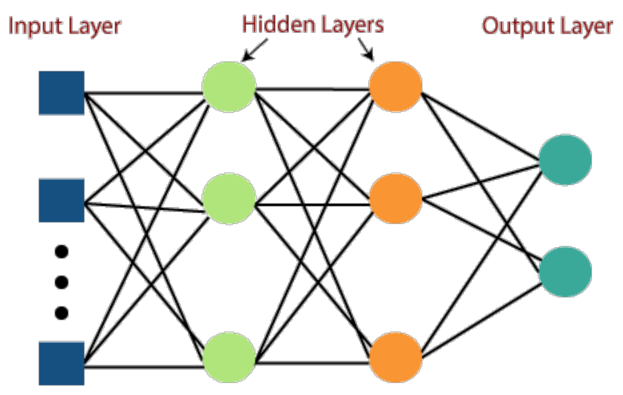


Fig. 4. Multilayer perceptron structure

The dataset it is chosen because for 25 years Ecuador has been the leading exporter of banana in the world. The banana trade represents for the country, after oil, the second source of income for its economy and, consequently, contributes significantly to the long process of its development [6]. However, in recent years, banana exports have undergone a variation, becoming the second product of non-oil exports [7]. According to [8] the exportation of this fruit until October 2020 were of 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.

III. Implementation of the Solution

The entire project is implemented on python 3.

A. Libraries

The principal libraries used for different proposes are:

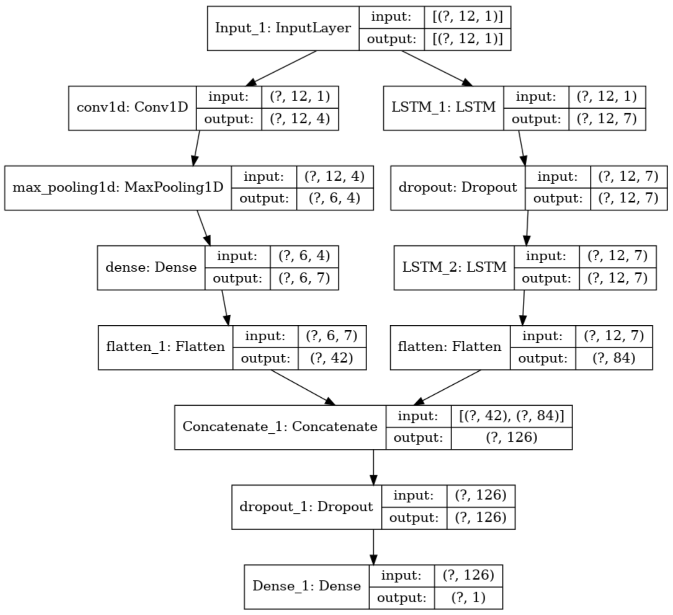
- Tensorflow and Keras (version 2): to plot, develop, assemble, training, and testing deep neural networks.
- Pandas: to process and manage large sets of data.

B. Dataset

The dataset contain 248 values. Each value represents the quantity of monthly banana exports, measured in metric tons, made by Ecuador from January 2000 to August 2020. It was taken from Banco Central del Ecuador database which it is update each month.

C. DNN models

Six DNN models are assembled. Three linear and non-linear models with several hidden layers; however, the non linear models are distributed in two columns. In each column it is found two different models which at the end are concatenated by just one layer. On Fig.5. it is appreciated the non linear model for a combination of CNN and LSTM.



Input_1: InputLayer | input: [(?, 12, 1)] | output: [(?, 12, 1)]
conv1d: Conv1D | input: (?, 12, 1) | output: (?, 12, 4)
LSTM_1: LSTM | input: (?, 12, 1) | output: (?, 12, 7)
max_pooling1d: MaxPooling1D | input: (?, 12, 4) | output: (?, 6, 4)
dropout: Dropout | input: (?, 12, 7) | output: (?, 12, 7)
dense: Dense | input: (?, 6, 4) | output: (?, 6, 7)
LSTM_2: LSTM | input: (?, 12, 7) | output: (?, 12, 7)
flatten_1: Flatten | input: (?, 6, 7) | output: (?, 42)
flatten: Flatten | input: (?, 12, 7) | output: (?, 84)
Concatenate_1: Concatenate | input: [(?, 42), (?, 84)] | output: (?, 126)
dropout_1: Dropout | input: (?, 126) | output: (?, 126)
Dense_1: Dense | input: (?, 126) | output: (?, 1)

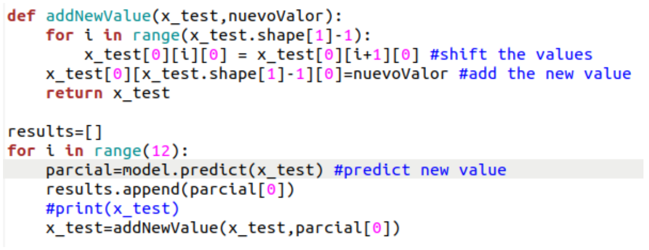
Fig. 5. Non-linear model for CNN-LSTM neural network

The linear models are assembled from the Sequential API on Keras, and the non-linear models are assembled from the Functional API on Keras which can handle models with non-linear topology.

D. Methodology

First, the dataset is processed and converted into a supervised learning. It is proved with a window size of 6,12 and 24.It means that we move around the values taking 6, 12 ans 24 moths as inputs and 1 month as output. Second, the neural networks are trained and evaluated to predict one month; however, once the network is ready, this is used in a for loop with the sliding window method to predict 12 moths Fig.6.

For the linear models the number of input, output, hidden and neurons (or LSTM cells for LSTM models) are fixed to 1, 1, 8 and 7, respectively for each model. Moreover, the dropout rate is fixed to 0.1 for all the models, and the maximun number of epochs is set to 800 . In the case of CNN 3 convolutions layers (with 4-8-12 filters and 7-5-3 kernel size) and 3 max pooling layers (with a pool size of 2 for all the layers). Finally, for the non-linear models the hyperparameters configuration is the same; however, the number layers is reduced. Once the model with less error values is found, it is used to make predictions for the next months. Finally, all the results are compared.



```
def addNewValue(x_test,nuevoValor):
    for i in range(x_test.shape[1]-1):
        x_test[0][i][0] = x_test[0][i+1][0] #shift the values
    x_test[0][x_test.shape[1]-1][0]=nuevoValor #add the new value
    return x_test

results=[]
for i in range(12):
    parcial=model.predict(x_test) #predict new value
    results.append(parcial[0])
    #print(x_test)
    x_test=addNewValue(x_test,parcial[0])
```

Fig. 6. Sliding Window in a for loop.

E. Metrics

- The use of Root Mean Squared Error (RMSE) is very common, and it is considered an excellent general-purpose error metric for numerical predictions [9]. RMSE is the standard deviation of the residuals (prediction errors). Residuals are a measure of how far from the regression line data points are, then RMSE is a measure of how spread out these residuals are around the line of best fit.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(S_i - O_i)^2}$$

Where O_i are the observations, S_i predicted values of a variable, and n the number of observations available for analysis.

IV. Results

	Window Size: 6					
	Epochs 800					
Model/Error	**Loos Train**	**Loos Test**	**RMSE Train**	**RMSE Test**	**RMSE Train**	**RMSE Test**
MLP	0.010	0.040	0.083	0.169	46995.17	93161.06
LSTM	0.011	0.053	0.087	0.20	49582.42	107037.33
CNN	0.009	0.043	0.079	0.176	45167.48	96215.48
MLP - LSTM	0.009	0.037	0.081	0.156	45601.62	89383.41
CNN - LSTM	0.008	0.039	0.076	0.165	43317.58	91919.80
CNN - MLP	0.009	0.030	0.079	0.138	45804.62	80703.85

TABLE I
LOSS AND RMSE FOR EACH MODEL WITH A WINDOW SIZE = 6

	Window Size: 12					
	Epochs 800					
Model/Error	**Loos Train**	**Loos Test**	**RMSE Train**	**RMSE Test**	**RMSE Train**	**RMSE Test**
MLP	0.008	0.017	0.077	0.108	43212.47	61832.31
LSTM	0.006	0.070	0.062	0.226	35976.25	122154.53
CNN	0.005	0.019	0.062	0.108	35744.58	63878.42
MLP - LSTM	0.006	0.010	0.064	0.076	36762.80	46699.41
CNN - LSTM	0.008	0.029	0.072	0.139	41507.42	79580.94
CNN - MLP	0.004	0.008	0.056	0.071	32584.83	43001.80

TABLE II
LOSS AND RMSE FOR EACH MODEL WITH A WINDOW SIZE = 12

V. Analysis of Results

Table I, II and III show that while the window size increase, the loss and RMSE for training and test dataset in all the

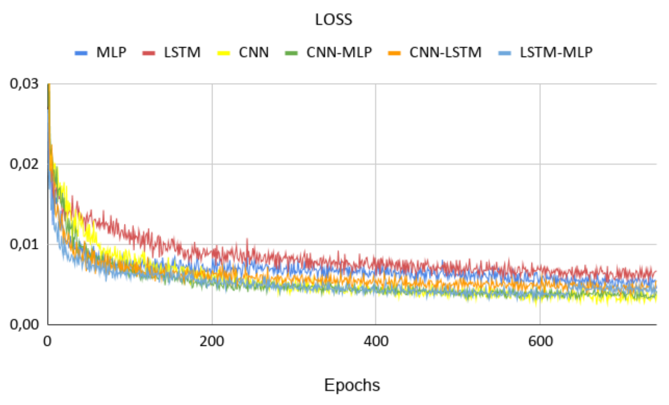


Fig. 7. Loss value across 800 epcohs

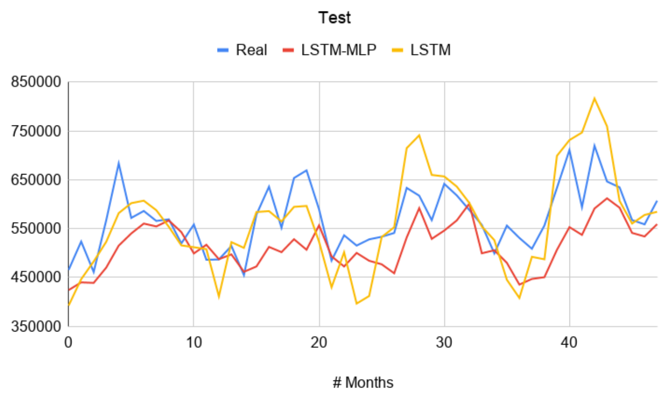


Fig. 10. LSTM-MLP vs LSTM

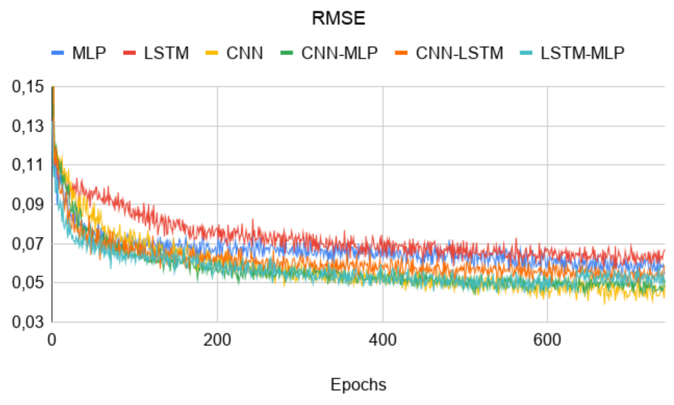


Fig. 8. RMSE value across 800 epochs

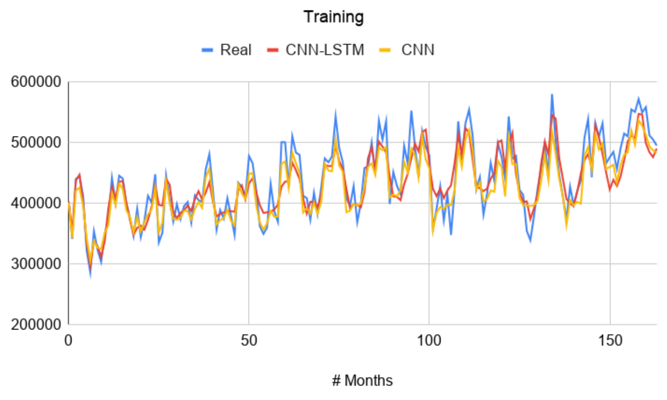


Fig. 11. CNN-LSTM vs CNN

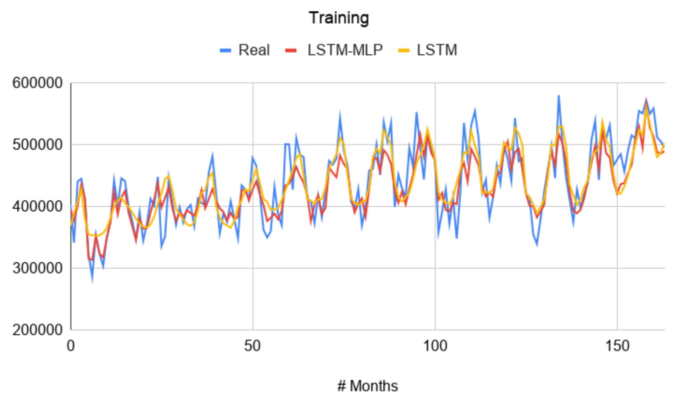


Fig. 9. LSTM-MLP vs LSTM

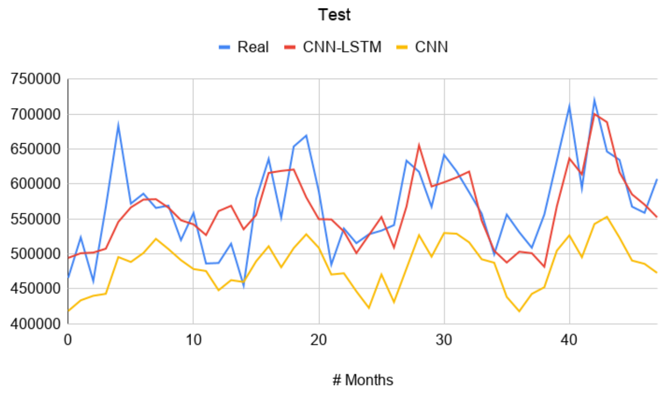


Fig. 12. CNN-LSTM vs CNN

	Window Size: 24					
	Epochs 800					
Model/Error	**Loos Train**	**Loos Test**	**RMSE Train**	**RMSE Test**	**RMSE Train**	**RMSE Test**
MLP	0.004	0.011	0.054	0.079	31256.45	49731.17
LSTM	0.005	0.022	0.056	0.118	33028.82	69279.90
CNN	0.002	0.022	0.043	0.128	24995.99	68911.04
MLP - LSTM	0.004	0.026	0.053	0.129	30134.83	74645.47
CNN - LSTM	0.003	0.015	0.045	0.096	26558.19	57177.65
CNN - MLP	0.003	0.009	0.044	0.075	25679.96	44955.96

TABLE III
LOSS AND RMSE FOR EACH MODEL WITH A WINDOW SIZE = 24

Month	CNN-MLP
9	541,527.53
10	555,770.85
11	538,186.75
12	548,936.11
1	556,637.15
2	595,146.80
3	648,318.07
4	616,288.74
5	671,796.67
6	651,032.14
7	631,098.72
8	585,326.92

TABLE IV
MONTHLY BANANA EXPORTS PREDICTED FROM 01/09/2020 TO 01/08/2021 WITH CNN-MLP MODEL

models decrease. It also shows that CNN have a value of 0.022 and 0.128 for loss-test and RMSE-test, respectively; however, on TableII 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 it is seem for the CNN-MLP and MLP-LSTM.

Fig.7 depicts how the loss value decrease for all the models during the train. It show that CNN and CNN-MLP have the lowest values (0.002 and 0.003 respectively, according to TableIII) and are almost overlapped from around 200 epochs. It is interesting see how MLP start with the lower loss values; however, from around 600 epochs it start to increase its values. In addition, Fig.7 shows that LSTM model have the highest

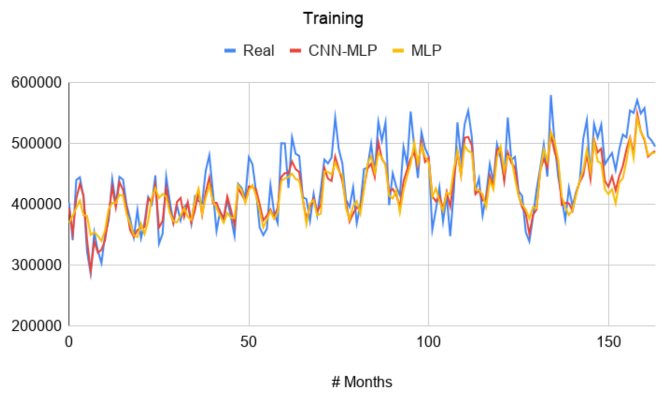


Fig. 13. CNN-MLP vs MLP

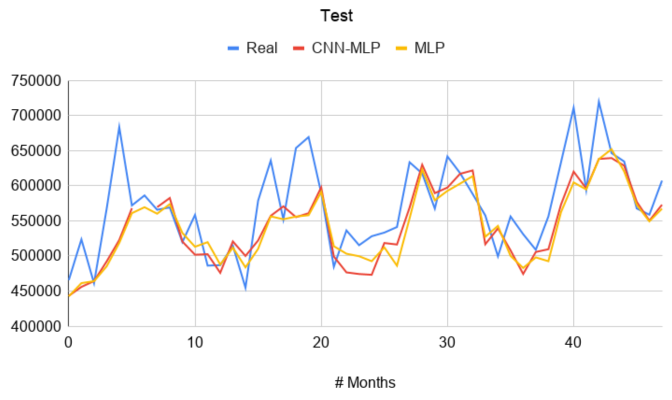


Fig. 14. CNN-MLP vs MLP

loss value, as it is also seem on Table III where its value is 0.005.

Fig.8 shows how the RMSE value decrease for all the models during the train. It shows that, for all the models, the RMSE values have a very similar behavior as on Fig.7; however, its values are higher. Then, CNN and and CNN-MLP have the lowest values (0.043 and 0.044 respectively) and LSTM has the higher value (0.056). These RMSE values can be also seen on TableIII

Fig.9 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 process more data than MLP and LSTM. Nevertheless, it is appreciated that when LSTM is combine with MLP the smooth disappear and the results improve. On Table III it is appreciated that the loss and RMSE value for LSTM are 0.005 and 0.056 respectively, and for MLP-LSTM 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.

Fig.10 shows the predicted values for the test dataset with LSTM and LSTM-MLP. The LSTM values from 20 month start to drift away while LSTM-MLP are more closer to the real ones. On Table III it is appreciated 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.

Fig.11 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 real values. On Table III it is appreciated 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. Then, CNN has better results on test.

Fig.12 shows the predicted values for the test dataset CNN and CNN-LSTM. The CNN-LSTM values are closer than CNN. On Table III it is appreciated that the loss and RMSE value 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.

Fig.13 shows the predicted values for the training dataset with MLP and CNN-MLP. Both of the models shows a good performance. It seems CNN improve the results at the beginning of the series, then from 50 month its values seems very similar. On Table III it is appreciated that the loss and RMSE value for MLP are 0.008 and 0.077 respectively, and for CNN-MLP are 0.004 and 0.056. Moreover the difference between the predicted values and the real ones of MLP and CNN-MLP is 31k and 25k, respectively. Then, CNN has better results on training.

Fig.14 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. On Table III it is appreciated that the loss and RMSE value for CNN are 0.022 and 0.128 respectively, and for CNN-MLP are 0.009 and 0.075. Moreover the difference between the predicted values and the real ones of MLP and CNN-MLP is 49k and 44k, respectively. Then, CNN-MLP has better results on test.

Finally, on Table IV it is show the values in mectric tons for next 12 months by using the model with less error obtained, which is LSTM-Par.

VI. HPC considerations

In this projects the time for loading the dataset is not long because it is one vector (1 x 248). However, if the vector size, the window size, and the number of layers on each network increase, then the execution time to process all the data, convert it into a supervised problem, and pass all this information through the neural networks may increase and it would be necessary to use some hpc techniques; such as loading data in parallel in order to decrease the execution time. Furthermore, when the neural networks are ready to work, the execution time increase because they will need to be trained, and depending on the number of epoch they will need more time and computational power.

One HPC enviroment that offers data parallelism for on Tensorflow/Keras is the Benchmark called: Mirrored Strategy which works with GPU's and if there is not GPU, it use all the CPU available as on GPU.

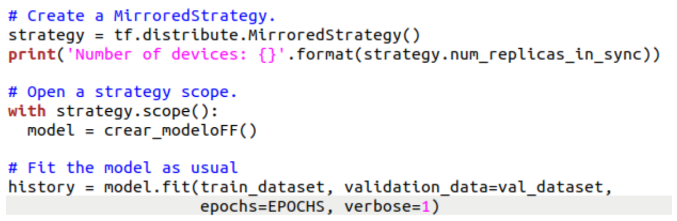
A. Mirrored Strategy

On a single machine with several GPU's on it. Each GPU runs a copy of the neural network model (called replica), then processes different batches of data on each of the GPU's and finally they merge the results. For instance, let's say there is a machine with 8 GPU's and at each step of training:

1) The current batch of data (called global batch) is split into 8 different sub-batches (called local batches). For example, if the global batch has 512 samples, each of the 8 local batches will have 64 samples.
2) Each of the 8 replicas independently processes a local batch: they run a forward pass, then a backward pass, outputting the gradient of the weights with respect to the loss of the model on the local batch.
3) The weight updates originating from local gradients are efficiently merged across the 8 replicas. Because this is done at the end of every step, the replicas always stay in sync

B. Mirrored Strategy - Implementation

This strategy is applied with a few lines of code. First, it is created the strategy. Then, the scope of the strategy is opened and inside this, the construction and compilation of the neural network model is included. Finally, it is fitted the model as usual Fig.15



```
# Create a MirroredStrategy.
strategy = tf.distribute.MirroredStrategy()
print('Number of devices: {}'.format(strategy.num_replicas_in_sync))

# Open a strategy scope.
with strategy.scope():
  model = crear_modeloFF()

# Fit the model as usual
history = model.fit(train_dataset, validation_data=val_dataset,
                    epochs=EPOCHS, verbose=1)
```

Fig. 15. Mirrored strategy for Data Parallelism

VII. Conclusions

Increase the window size improves the RMSE and loss values. However, for some models such as CNN, CNN-MLP and MLP-LSTM these values start to increase instead of drecrease. Moreover, increase the window size many steps in time, may reframed the series until reach a point where there not exist enough information to process and learn by the neural networks.

Increase the window size means increase the number of inputs neurons in the input layer, this consequently also increase the time and number parameters or weigths for training.

Non-linear typologies helps to combine the capacity of inference of two or more deep neural networks by adding different types of layers (such as CNN, LSTM and Dense) in a parallel way in order to obtain better results and improve its accuracy.

The model with the higher loss and RMSE values is LSTM. Since LSTM process more parameters and retain information for longer periods than the rest of models it may need more epochs of training to improve its results.

The model with the lower loss and RMSE values is CNN-MLP. On the one hand, MLP overcomes CNN with its results for the training data set. On the other hand, CNN overcomes MLP with its result for testing data set. Therefore, by combining both of them, CNN-MLP improves its accuracy and overcomes the results of the individual models.

It is necessary be care full when new layers are added because it means more parameter of training, specially in the MLP that face the vanish gradient problem which makes it really hard to learn and tune the parameters of the earlier layers when the number of layers in the neural network increase.

Mirrored Strategy is a excellent benchmark used to make Data Parallelism with just include a few lines in the original code. It also can implement Model Parallelism, more information can be found on [11]

REFERENCES

[1] Liu, F., & Deng, Y. (2019). A fast algorithm for network forecasting time series. IEEE Access, 7, 102554-102560.

[2] Siami-Namini, S., Tavakoli, N., & Namin, A. S. (2018, December). A comparison of ARIMA and LSTM in forecasting time series. In 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA) (pp. 1394-1401). IEEE.

[3] Sit, M., & Demir, I. (2019). Decentralized flood forecasting using deep neural networks. arXiv preprint arXiv:1902.02308.

[4] Rangapuram, S. S., Seeger, M. W., Gasthaus, J., Stella, L., Wang, Y., & Januschowski, T. (2018). Deep state space models for time series forecasting. In Advances in neural information processing systems (pp. 7785-7794).

[5] S. Abirami, P. Chitra, in Advances in Computers, 2020

[6] Vásquez, R. (2017). El impacto del comercio del Banano en el desarrollo del Ecuador. AFESE.

[7] Pérez, J. (2018). Estudio de factibilidad para el establecimiento de una exportadora de banano en Guayaquil, Ecuador, para su comercialización en Alemania. (Tesis de Grado). Repositorio Escuela Agrícola Panamericana Zamorano.

[8] Alvarado, P. (2020). Inversiones se reflejan en un mejor desempeño bananero. Revista Líderes

[9] Simon P. Neill, & M. Reza Hashemi.(2018) Fundamentals of Ocean Renewable Energy. E-Business Solutions

[10] Adetiloye, T., & Awasthi, A. (2017). Predicting Short-Term Congested Traffic Flow on Urban Motorway Networks. In Handbook of Neural Computation (pp. 145-165). Academic Press.

[11] Chollet, F. (2020).Guide to multi-GPU & distributed training for Keras models. Retrieve from: https://keras.io/guides/distributed_training/