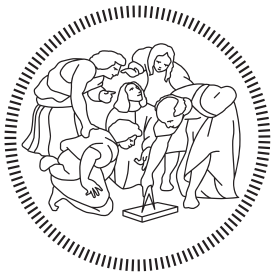
AY 2021/2022



Politecnico di Milano

Numerical Analysis for Machine Learning

Bitcoin Price Prediction

Group 22

Piero Rendina (10629696)
Andrea Sanchini (10675541)

Professor
Edie Miglio

March 2, 2022

Contents

1 Introduction

Cryptocurrencies are digital currencies secured by cryptography protocols. They overcome many problems affecting traditional currency like forgery, double spending, ownership. Moreover, digital currencies fulfil the money exchange among two people without relying on any intermediary. The transactions are verified and stored by a decentralized network of computers without distinction.
Nowadays, their employment has many applications ranging from buying goods and services to making profitable investments. They rely neither on any government authority nor on central banks, resulting in the absence of laws. The lack of regulation entails significant volatility and strong price fluctuations over time. For this reason, a considerable amount of people focuses their effort on finding a model capable of predicting the behaviour of these assets.
In general, time-series forecasting is one of the most challenging fields in the scope of machine learning. Furthermore, performing predictions in a domain with unknown trends and many unpredictable factors hampers the task even more.
That being said, the project aims to address the problem of forecasting the Bitcoin price evolution exploiting recurrent neural networks, more specifically **LSTM** (long short term memory) models.

2 Technology

The project employs an extension of Recurrent Neural Network models.
The Recurrent Neural Network (**RNN**) is a class of neural networks well-suited to analyze time series. It enhances the feedforward network capabilities by adding backward connections to retain information (as shown in the figure 1). Thus, the output of a recurrent neuron at time step t is a function of the current input, combined with its state.

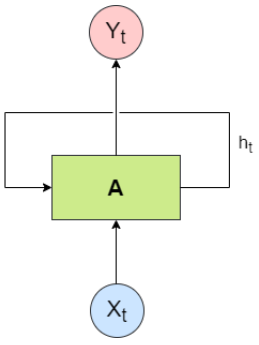


Figure 1: *Compact representation of a RNN*

- $\mathbf{Y_t}$ is the output at time step t
- $\mathbf{X_t}$ is the input at time step t
- $\mathbf{h_t}$ represents the memory of the network at time step t.

Due to its simplicity, a recurrent cell is only capable of learning short-term patterns, meaning that it struggles to understand relationships over longer time frames.
As mentioned before, the main drawback deals with handling the long-term dependencies.
New classes of neurons arise to tackle this issue. One of them, the most popular, is the LSTM (long short term memory) cell. The key idea is to let the network understand what to store in the long-term memory, what to throw away and what to read from it.

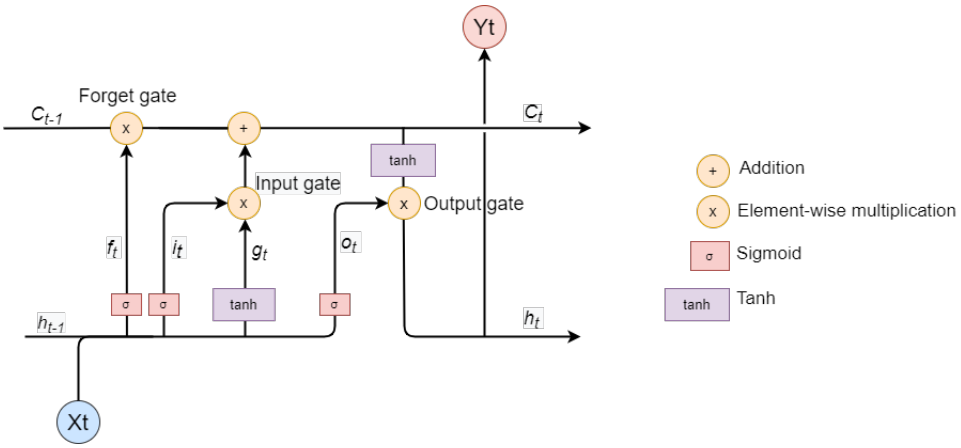


Figure 2: *Representation of a LSTM cell*

The cell exploits the sigmoid activation function:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

whose output ranges from 0 to 1, to create gate controllers.

The following is a brief description of the signals and the states.

- $\mathbf{c_t}$ is the long-term state (6)
- $\mathbf{h_t}$ is the short-term state (7)
- $\mathbf{i_t}$ signal for the input gate. It controls which parts of the $\mathbf{g_t}$(5) should be added to the long-term state (2)
- $\mathbf{f_t}$ signal for the forget gate. It controls which parts of the long-term state should be erased (3)
- $\mathbf{o_t}$ signal for the output gate. It controls which parts of the long-term state should be read and output at this time step (4)

The equations that summarize the behaviour of the cell for a single instance are shown below

$$i_{(t)} = \sigma(W_{xi}^T \times x_{(t)} + W_{hi}^T \times h_{(t-1)} + b_i) \tag{2}$$

$$f_{(t)} = \sigma(W_{xf}^T \times x_{(t)} + W_{hf}^T \times h_{(t-1)} + b_f) \tag{3}$$

$$o_{(t)} = \sigma(W_{xo}^T \times x_{(t)} + W_{ho}^T \times h_{(t-1)} + b_o) \tag{4}$$

$$g_{(t)} = \tanh\left(W_{xg}^T \times x_{(t)} + W_{hg}^T \times h_{(t-1)} + b_g\right) \tag{5}$$

$$c_{(t)} = f_{(t)} \otimes c_{(t-1)} + i_{(t)} \otimes g_{(t)} \tag{6}$$

$$y_{(t)} = h_{(t)} = o_{(t)} \otimes \tanh c_{(t)} \tag{7}$$

3 Dataset overview

The dataset is drawn from the Yahoo[1] API and it provides data concerning the daily bitcoin price evolution. Each day represents a row in our table with eight features. The features are:

1. **date**: day of the observation
2. **opening price**: price at the moment of stock market opening
3. **highest price**: highest price reached throughout the day
4. **lowest price**: lowest price reached throughout the day
5. **closing price** price at the moment of stock market closing
6. **adjusted closing price**: it amends the Bitcoin closing price to reflect that stock's value after accounting for any corporate actions like dividends, stock splits
7. **volume**: it is the overall amount of bitcoin traded in one day

The prices are expressed in US dollars and the entries look like the following ones

Date	Open	High	Low	Close	Adj Close	Volume
2022-02-19	40022.132	40407.480	39724.722	40165.738	40165.738	1.6×10^{10}
2022-02-20	40118.101	40119.890	38112.812	38431.378	38431.378	1.8×10^{10}
2022-02-21	38423.210	39394.437	36950.476	37075.281	37075.281	2.9×10^{10}

[1] https://finance.yahoo.com

Figure 3: *Evolution of the Bitcoin price over the last three months*

4 Data processing

4.1 Data split

The first step in the data preparation phase consists in splitting the dataset into a training set and a test set. The split schema we applied was 70% training data, 20% validation and 10% testing. The function below divides the data by putting the entries related to the current year in the test set to reach the 10% threshold. The validation and training partitions are managed by the keras fit[2] function.

```
def split_dataset(data = data, date = '2022-01-01'):
  df_train = data[data['Date']<date].copy()
  df_test = data[data['Date']>=date].copy()
  df_train = df_train.drop(['Date'], axis = 1)
  df_test = df_test.drop(['Date'], axis = 1)
  return df_train, df_test
```

4.2 Data normalization

The data is normalized according to the mean and the standard deviation drawn from the training set so that the models have no access to the values in the validation and test sets.

[2]https://keras.io/api/models/model_training_apis/#fit-method

4.3 Data windowing

The LSTM models expect the data to be in a matrix with dimensions: **[samples, time steps, features]**. The datasets are bidimensional, so we need to convert the sequences **[time steps, features]** into smaller overlapping windows and create a nested dataset. Each sample represents a window containing a fixed number of observations with the respective features' values and the associated label. The function presented below recalls the one presented in the article [3].

```
def split_sequence(sequence, window_dim, forecast_horizon,
                   full_featured = False, default_idx = 3):
  X, y = list(), list()
  for i in range(len(sequence)):
    input_end = i + window_dim
    forecast_end = input_end + forecast_horizon
    if forecast_end > len(sequence):
      break
    if full_featured == True:
      seq_x, seq_y = sequence[i:input_end],
                     sequence[input_end:forecast_end]
    else:
      seq_x, seq_y = sequence[i:input_end],
                     sequence[input_end:forecast_end, default_idx]
    X.append(seq_x)
    y.append(seq_y)
  return np.array(X), np.array(y)

X, y = split_sequence(data_train_norm, window_dim=7, forecast_horizon=1)
```

5 Models

We developed several models belonging to two different classes to address the bitcoin trend forecast: **sequence-to-sequence** and **sequence-to-vector** models. They were thought to predict the closing price of Bitcoin corresponding to feature n.5 with regard to the dataset overview.

5.1 Sequence-to-vector models

This class of models is fed with data over the last N observations, and it must output the closing price for the time step $N+1$ as shown in the fig. 7

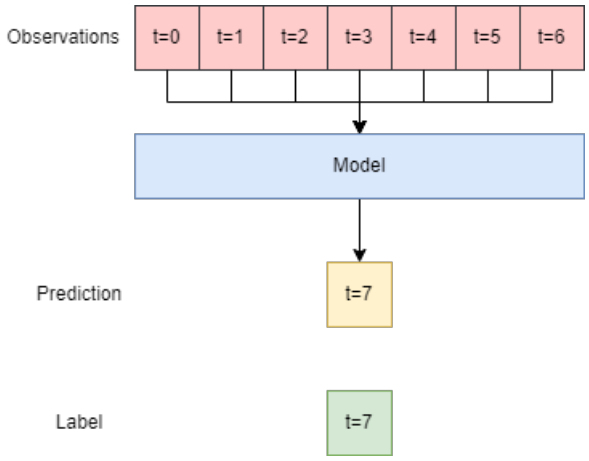


Figure 4: *The model exploits the last seven time steps to come up with a prediction over the eighth time step.*

5.1.1 Baseline model

This model is introduced as a benchmark for the evaluation of more complex solutions as suggested in the tutorial linked here [2]. The price predicted at time step t is given by the time step t-1.

```
def baseline_model(input):
  output = np.array([input[i] for i in range(0,len(input)-1,1)])
  return output
```

5.1.2 Pure LSTM model

The model is the simplest implementation of the LSTM recurrent network, constituted by just one layer with 50 neurons. What's worth noticing is the argument **return_sequence** set to false. It entails that whenever the network elaborates a time-steps window, it exploits the first N-1 samples to warm up and the last one to output the price.

```
model = Sequential()
model.add(LSTM(units = 50, activation = 'tanh',
            return_sequences = False,
            input_shape = (X_train.shape[1], X_train.shape[2])))
model.add(Dropout(0.2))
model.add(Dense(units = 1))
```

5.1.3 CNN-LSTM model

This model proposed is drawn from the paper [4]. The implementation is carried out through the API provided by https://keras.io/api/models/.
Below the code to build the model.

```
model = Sequential()
model.add(Conv1D(filters = 32, kernel_size = 1, activation = 'tanh',
```

```
          padding = 'same', input_shape = (X_train.shape[1], X_train.shape[2])))
model.add(MaxPooling1D(pool_size=1, padding = 'same'))
model.add(LSTM(64, activation='tanh',
          input_shape=(X_train.shape[1], X_train.shape[2])))
model.add(Dropout(0.2))
model.add(Dense(units = 1))
```

5.1.4 Multi-currency CNN-LSTM model

This model is drawn from the paper [5]. It works with two cryptocurrencies (Ethereum, Ripple) other than Bitcoin to come up with hidden correlations.

```
input_shape_1 = Input(shape=(7, 6))
input_shape_2 = Input(shape=(7, 6))
input_shape_3 = Input(shape=(7, 6))
crypto_1 = Conv1D(filters = 16, kernel_size = 2)(input_shape_1)
crypto_1 = MaxPooling1D(pool_size=2)(crypto_1)
crypto_1 = LSTM(50)(crypto_1)
model_1 = tf.keras.Model(inputs=input_shape_1, outputs=crypto_1)

crypto_2 = Conv1D(filters = 16, kernel_size = 2)(input_shape_2)
crypto_2 = MaxPooling1D(pool_size=2)(crypto_2)
crypto_2 = LSTM(50)(crypto_2)
model_2 = tf.keras.Model(inputs=input_shape_2, outputs=crypto_2)

crypto_3 = Conv1D(filters = 16, kernel_size = 2)(input_shape_3)
crypto_3 = MaxPooling1D(pool_size=2)(crypto_3)
crypto_3 = LSTM(50)(crypto_3)
model_3 = tf.keras.Model(inputs=input_shape_3,
                         outputs=crypto_3)

merged = tf.keras.layers.concatenate([model_1.output, model_2.output, model_3.output],
                                     axis=1)

out = Dense(64)(merged)
out = Dropout(0.3)(out)
out = Dense(32)(out)
out = Dropout(0.2)(out)
out = Dense(1)(out)

model = tf.keras.Model(inputs=[model_1.input, model_2.input, model_3.input],
                       outputs=out)
```

5.1.5 Training

We employed several training options, involving the choice of the optimizer, the activation functions and the number of neurons per layer as well as the size of the input window. Eventually, the training phase was carried out with the following configuration:

- **number of epochs**: 300

- **optimizer**: adam[3] with **batch size**: 50
- **activation function**: *tanh*
- **callback**: early stopping on the validation loss

```
callback = tf.keras.callbacks.EarlyStopping(
monitor="val_loss",
min_delta=0.0005,
patience=20,
mode="min"
)
```

- **loss function**: mean squared error

5.1.6 Performance

The investments in financial markets are linked to how the stock behaves after the purchase or the sale. From an investor point of view, we are more focused on whether the stock value will go up or down. Therefore, we extended the concept of accuracy to time-series prediction problems. In this domain, accuracy is the percentage of trends correctly predicted. We defined a trend as a price increase or decrease among two consecutive days.
Here is the definition:

$$Accuracy = \frac{(TP + TN)}{(TP + TN + FP + FN)} \tag{8}$$

where:

- **TP** stands for the number of correctly identified increases
- **TN** stands for the number of correctly identified decreases
- **FP** stands for the number of trends which were misidentified as a price increase
- **FN** stands for the number of trends which were misidentified as a price decrease

These are the results obtained:

[3] https://keras.io/api/optimizers/adam/

Model	Lag	Dropout	Accuracy	RMSE	MAE	MFE
Baseline	1	-	47.82%	1289.56	901.35	160.88
Pure LSTM	7	0	52.17%	2041.91	1572.79	150.92
	7	0.1	53.62%	2155.63	1672.26	944.05
	7	0.2	52.17%	1954.67	1522.41	574.56
	14	0	47.82%	2591.72	2069.04	742.67
	14	0.1	52.17%	1845.38	1407.17	574.03
	14	0.2	52.17%	1973.37	1576.50	250.38
	30	0	53.62%	2188.64	1721.89	716.93
	30	0.1	50.72%	2145.70	1737.19	110.07
	30	0.2	50.72%	2167.48	1714.14	835.10
CNN-LSTM	7	0	55.71%	1698.63	1329.49	-169.69
	7	0.1	54.29%	1591.99	1207.39	-46.11
	7	0.2	52.86%	1734.65	1327.92	630.99
	14	0	54.29%	1935.12	1520.73	99.33
	14	0.1	54.29%	1837.41	1384.44	780.13
	14	0.2	54.29%	2255.20	1734.78	1414.14
	30	0	55.71%	1893.10	1526.60	-709.03
	30	0.1	51.43%	1705.73	1289.67	465.24
	30	0.2	52.86%	1477.53	1144.92	-42.25
multi CNN-LSTM	7	0.2	50.00%	2539.54	2087.15	1134.17
	14	0.2	50.00%	3027.32	2389.86	1809.60
	30	0.2	51.43%	2675.25	2130.75	1237.79

Metrics:

- **Lag**: dimension of the window shown in fig.7
- **Dropout**: specified rate for the dropout layer, which randomly sets input units to 0 to avoid overfitting
- **RMSE**: stands for root mean squared error

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(Y_{pred,i} - Y_i)^2}{N}} \tag{9}$$

- **MAE**: stands for mean absolute error

$$MAE = \frac{\sum_{i=1}^{N}\left|(Y_{pred,i} - Y_i)\right|}{N} \tag{10}$$

- **MFE**: stands for mean forecast error

$$MFE = \frac{\sum_{i=1}^{N}(Y_{pred,i} - Y_i)}{N} \tag{11}$$

5.2 Sequence-to-sequence models

These models exploit the last N prices to output the prices starting from the first day in the future over a fixed time frame as shown in (fig. 6)

5.2.1 Pure LSTM model

The model is a simple implementation constituted by two LSTM layers with 50 neurons each, followed by a Dense layer as proposed in[1].

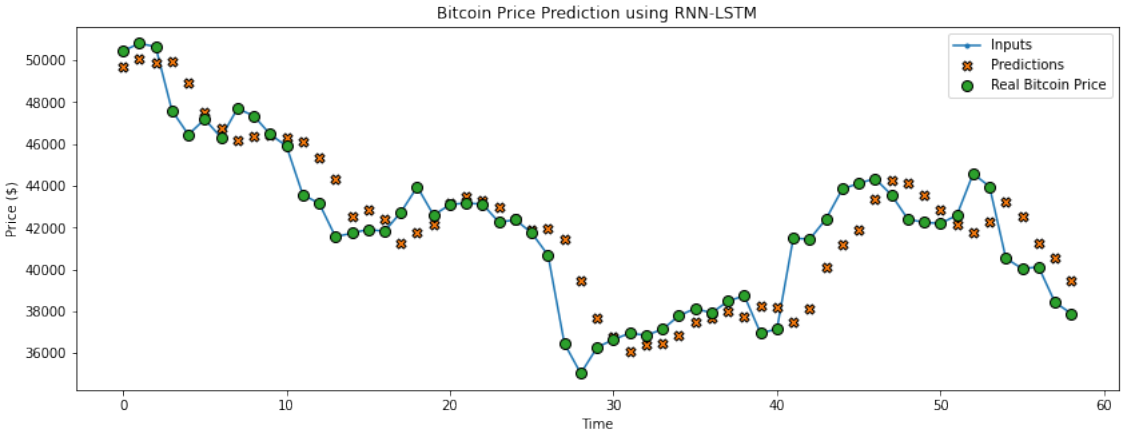


Figure 5: *Example of prices predicted by the single step model.*

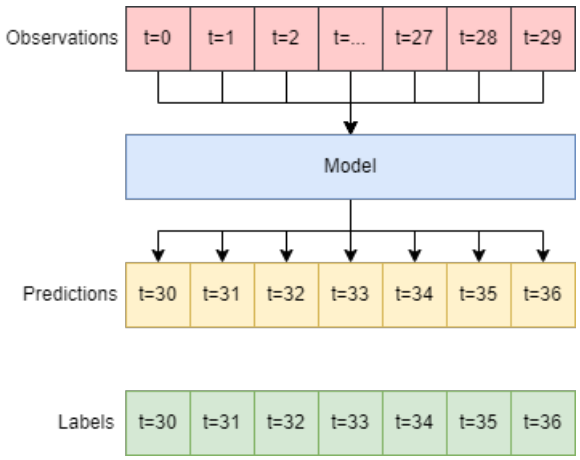


Figure 6: *The model works with thirty time steps windows and makes predictions over a seven time steps horizon.*

```
multi_step_model = Sequential()
multi_step_model.add(LSTM(units = 50, activation = 'tanh', return_sequences = True,
                     input_shape = (X_multi_step.shape[1], X_multi_step.shape[2])))
multi_step_model.add(Dropout(0.1))
multi_step_model.add(LSTM(units = 50, activation = 'tanh', return_sequences = False))
multi_step_model.add(Dropout(0.2))
multi_step_model.add(Dense(32))
multi_step_model.add(Dropout(0.2))
multi_step_model.add(Dense(y_multi_step.shape[1]))
```

5.2.2 CNN-LSTM model

The model is equivalent to the one described in section 5.1.3, with slight modifications in order to perform multi-step predictions.

```
multi_step_model.add(Conv1D(16, kernel_size=2,
                     input_shape=(X_multi_step.shape[1], X_multi_step.shape[2])))
multi_step_model.add(AveragePooling1D(pool_size=2))
multi_step_model.add(LSTM(50, activation='tanh',
                     input_shape=(X_multi_step.shape[1], X_multi_step.shape[2])))
multi_step_model.add(BatchNormalization())
multi_step_model.add(Dropout(0.2))
multi_step_model.add(Dense(32))
multi_step_model.add(BatchNormalization())
multi_step_model.add(Dropout(0.2))
multi_step_model.add(Dense(y_multi_step.shape[1]))
```

5.3 Training

The training was carried out with same options described in section 5.1.5.

5.3.1 Performance

To evaluate the performance of the two models we employed the same metrics described in section 5.1.6. These are the results obtained:

Model	Lag	RMSE	MAE	MFE
Pure LSTM	7	3101.02	2465.85	781.99
	14	3650.67	2972.62	1395.42
	30	3650.44	3028.29	186.85
CNN-LSTM	7	3136.75	2477.35	1332.45
	14	3289.00	2657.15	329.61
	30	3668.51	3006.75	-1461.24

The multi step models required a different approach in the accuracy computation. The metric is assessed on each day of the forecast horizon. Our objective was to evaluate whether the predictions are more precise on the earliest days of the time range.

Model	Lag	1st trend	2nd trend	3rd trend	4th trend	5th trend	6th trend
Pure LSTM	7	52.11%	47.88%	54.92%	47.88%	47.88%	52.11%
	14	48.43%	51.56%	50.00%	48.43%	51.56%	48.43%
	30	50.00%	47.91%	45.83%	50.00%	47.91%	43.75%
CNN-LSTM	7	59.15%	54.92%	40.84%	47.88%	53.52%	52.11%
	14	51.56%	45.31%	50.00%	51.56%	50.00%	56.25%
	30	50.00%	43.75%	50.00%	50.00%	50.00%	47.91%

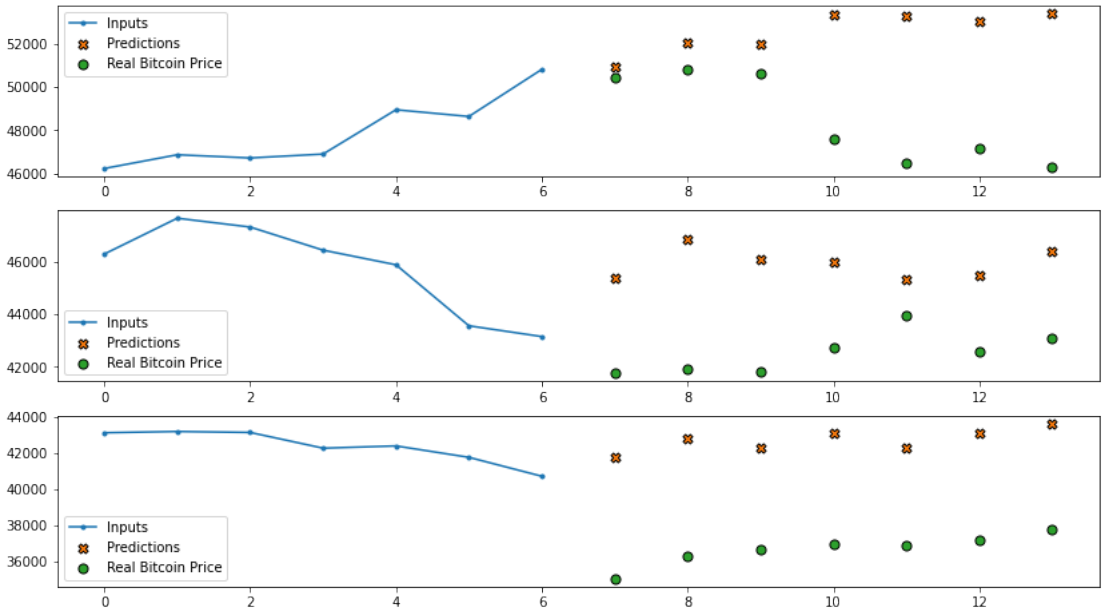


Figure 7: *Example of prices predicted by the multi step model.*

6 Conclusions

The conclusions we've reached throughout the project may not seem astonishing, as expected, considering the high volatility of the currency value coupled with external phenomena not taken into account by the networks. On average, the models overestimate the price prediction. The best accuracy value is 55.71%, barely above what we would expect with a random trend forecast (where 50% is the probability of going up/down). Nevertheless, the accuracy improves with respect to the baseline model in section 5.1.1, whose value is 47.82%, being greater than 50%. The single currency models work better with 7-14 time steps windows, whereas the multi currency one is more accurate with larger windows. The RMSE is affected by the wide price fluctuations, emphasized over the last year. On average, the CNN-LSTM model described in section 5.1.3 provides the lowest RMSE value.
Concerning the multi step models, their behaviour is less accurate, therefore we could not gain meaningful results. Improvements may be possible by making the data stationary or leveraging autoregressive integrated moving average models (**ARIMA** models).

References

[1] URL: https://www.tensorflow.org/tutorials/structured_data/time_series.

[2] Jason Brownlee. URL: https://machinelearningmastery.com/time-series-forecasting-long-short-term-memory-network-python/.

[3] Halil Hertan. URL: https://towardsdatascience.com/cnn-lstm-based-models-for-multiple-parallel-input-and-multi-step-forecast-6fe2172f7668.

[4] Wenjie Lu; Hassanien Abd E.I.; Jiazheng Li; Yifan Li; Sun Aijun; Wang Jingyang; "A CNN-LSTM-Based Model to Forecast Stock Prices". In: (). URL: https://doi.org/10.1155/2020/6622927.

[5] Livieris L.E.; Kiriakidou N; Stavroyianni N.; Pintelas P; "An Advanced CNN-LSTM Model for Cryptocurrency Forecasting". In: *Electronics 2021, 10, 287* 9 (). URL: https://www.mdpi.com/2079-9292/10/3/287.