# Machine Learning algorithms applied to fluid dunamics: prediction of a flow around a cylinder

Anna Nava 12109889, Marta La Franca 12108945, Yousef Bahri 12111057

*Alpen-Adria-Universität Klagenfurt*

**Abstract**

Despite of its simple geometry, the fluid flow around a circular cylinder can request numerical solutions of significant computational costs. For this reason, the application of Machine Learning and Deep Learning algorithms to this problem is of great interest, since it can potentially be a good alternative to provide faster and better solutions.
The project goal is to predict the trend of this flow. The data utilised contains numerical solutions of the problem: it provides values of pressure and velocity of 11160 points, taken each 10 second during an overall time of 10000 seconds. For our purpose, Recurrent Neural Networks models - in particular LSTM (Long-Short Term Memory) and CNN (Convolutional Neural Network) - are used.
Subsequently, various metrics are applied to evaluate the quality of the predictions.

*Keywords:* LSTM, RNN, CNN, circular cylinder, Sequence.

## 1. Introduction

In the research field of fluid mechanics, the flow around a circular cylinder is one of the most classical problem and it has always been an important issue. Therefore, it's of fundamental importance for those who work in fluid dynamics being able to develop models to estimate the trend of the flow. For this reason, numerous predictive models have been proposed in the literature to solve similar kind of problems (e.g. [1], [2]).

In this project, we aim to propose further competitive Machine Learning predictive models for this topic. In particular, in this work, we have data which concerns 11160 points associated with their correspondent pressure and velocity components at specific times: our models are built to predict the flow of each point.

First of all, after splitting the data into train and test sets, it is also divided into sequences. Then, it is decided to apply a LSTM and a CNN network to predict the flow. In particular, eight distinctive prediction models are used, respectively two for the pressure and six for the components $U_x$, $U_y$ and $U_z$ of the velocity.
Afterward, the evaluation of the model based on different metrics is developed to verify the goodness of the models.

The work is organized as follows:

In the Sect. 2, the dataset available is described, highlighting the characteristics of the attributes.

The Sect. 3 focuses on the activities carried out to filter and ensure good data quality for the following analyzes.

In the Sect. 4, the problem is discussed and analyzed, specifying its goal.

The Sect. 5 contains a mathematical description of the chosen models, explaining their details and the reason why it was decided to use them.

Furthermore, in the Sect. 6, the algorithm performance and model evaluation are discussed.

In the end, the Sect. 7 explores the results obtained and describes their meaning.

The paper ends with our concluding notes in the Sect. 8.

## 2. Data description

The datasets of interest contain the numerical solution of flow around a cylinder with pressure and velocity components $(U_x, U_y, U_z)$. In particular, there are 1000 CSV files which are about data taken at the time interval of 10 seconds, along an overall time of 10000 seconds. Each file consists of 11160 rows and 7 columns, which describe the following quantities:

- P (Float): Pressure;
- U:0 (Float): $x$-component of velocity;
- U:1 (Float): $y$-component of velocity;
- U:2 (Float): $z$-component of velocity;
- Points:0 (Float): $x$-coordinate of the point;
- Points:1 (Float): $y$-coordinate of the point;
- Points:2 (Float): $z$-coordinate of the point.

Note that the coordinates at columns 4-7 are constants in all files: in fact, each of them contains the values of pressure and velocity for the fixed set of 11160 points at the time $t$. Moreover, the $z$-coordinates of the points and of the velocity are always equal to zero, so actually it's a problem that concerns only two dimensions ($x$ and $y$). Nevertheless, we've chosen to consider the third dimension, but it is negligible in this case. In the following image it is possible to observe how the problem appears in an instant of time $t$:

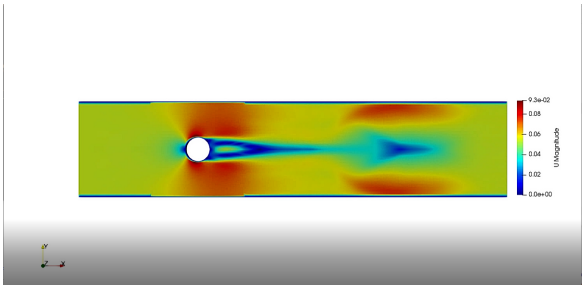


Figure 1: Screen of the problem at the time $t$

## 3. Data selection and preparation

First of all, three overall datasets are built, unifying all the files:

- `tot_pressure.csv`, which has on each column $j$ the values of pressure at the second $j$ for each point
- `tot_velocity_x.csv`, which has on each column $j$ the values of $U_x$ at the second $j$ for each point
- `tot_velocity_y.csv`, which has on each column $j$ the values of $U_y$ at the second $j$ for each point
- `tot_velocity_y.csv`, which has on each column $j$ the values of $U_z$ at the second $j$ for each point

They all have 11160 rows, one for each point, and 1001 columns (from second 0 to 10000). In this way, the target variables are studied separately.

At this point, it's possible to notice that the datasets are already clean, indeed there aren't any missing values or relevant outliers. Moreover, all pressure and velocity data belongs to a very restricted interval of values $I$ (where, to give an idea, $I \subset [-1/5, 1/5]$). For

this reason, data pre-processing techniques - like filling missing values, removing noise or data normalization - aren't necessary.

Initially, we split the datasets into training set and test set (respectively 80% and 20% of the overall dataset). Then, we still need to prepare data for the chosen models. In particular, our choice falls on **LSTM** and **CNN** networks, because of their ability to see patterns far back in time series data. The studied problem requires a prediction of 11160 parallel series (one for each considered point) and of multiple time steps into the future: it can be referred to as *multi-step time series forecasting*. For this goal, we split our multivariate sequences into samples with input and output components, following what is suggested in [8]. After having fixed a number of time steps *n_steps*, we reshape input data subdividing the time series into samples of *n_steps* consequentially instances, while the outputs are given by a single time step for each parallel series. Since an example is much more effective, let's see the practical case below, where the first sequences are the inputs and the second the outputs:

```
1        [[1  10  2]
2         [2  20  4]
3         [3  30  6]] [4  40  8]
4        [[2  20  4]
5         [3  30  6]
6         [4  40  8]] [5  50  10]
7        [[3  30  6]
8         [4  40  8]
9         [5  50  10]] [6  60  12]
```

To choose the number of time steps, we've made a trade off between the computational costs and the quality of the model. Since with too high *n_steps*, the computational time is really long, we've decided to fix *n_steps*=75.

## 4. Problem Analysis

### *4.1. Data exploration*

First of all, with a fast look at the dataset, we've noticed, as already said, that there are no missing or relevant anomalous values. Then, since we are interested in two different variables (pressure and velocity), we explore data about them separately.

Let's start to focus on pressure. At second zero, the pressure values associated to all the points are null. Then, the first thing that catches our attention is the fact that the following two values, corresponding to the seconds 10 and 20, are significantly bigger (in absolute value) to the other values for almost all considered points (see fig. 2). That can be explained by the fact that probably, at the beginning of the observations, a stronger pressure has been applied to the system. Anyway, these data values shouldn't effect significantly the predictions.

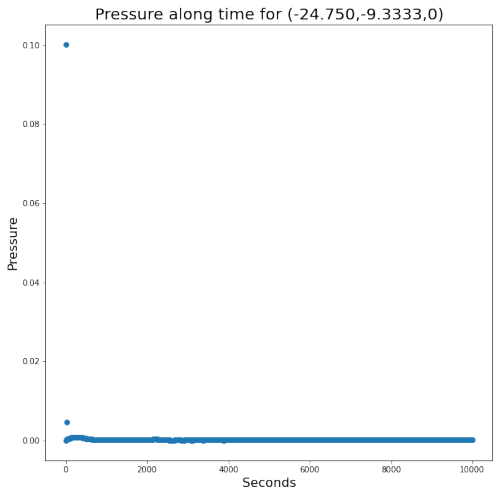


Figure 2: Values of pressure during the first seconds are outliers

Ignoring the first seconds, it's possible to notice that, for many points, the pressure, after a period of stabiliza-

tion, seems to assume a quite regular oscillating trend (see fig. 3).

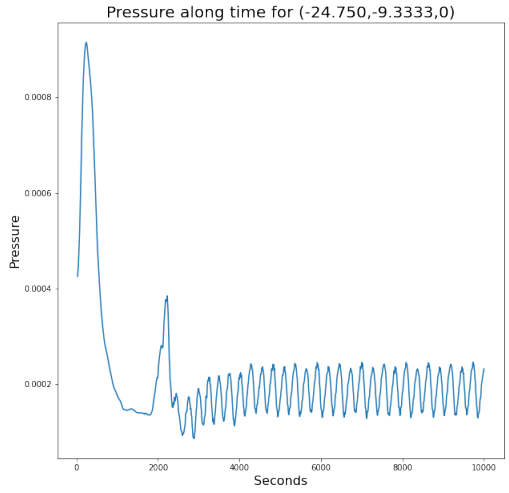


Figure 3: Trend of pressure ignoring the first seconds

Finally, let's see some statistics about the overall dataset of pressure `tot_pressure.csv` in the tab. 4.

Since we've observed that normalizing data doesn't

| | |
|---|---|
| mean | 0.000319 |
| std | 0.003166 |
| min | 0.000000 |
| 25% | 0.000148 |
| 50% | 0.000184 |
| 75% | 0.000226 |
| max | 0.100190 |

Figure 4: Summary statistics about pressure

effect significantly the results, we've decided to keep them as they are: in fact, they already have good statistical characteristics.

We can now shift our focus on the velocity. As already said, the component $z$ is always equal to zero, so, we are not interested in the third dimension. As observed for pressure, at second 0, the speed is null for every point. Then, for almost every points, velocity module reaches its maximum values for the first seconds, while, during the last seconds, it tends to go to zero. For instance, look at the graphic 5, where the dots indicate the extremities of the velocity vectors. In this representation, vectors aren't used because, otherwise, the result would have been really messy and hardy understandable.

In the end, summary statistics about the overall datasets

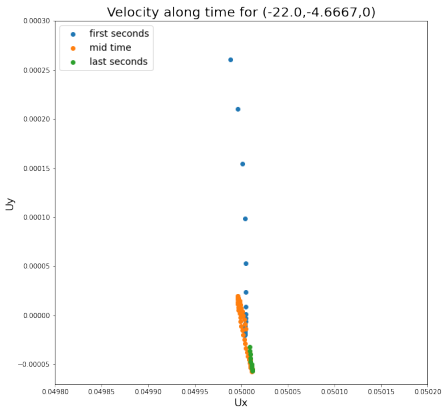


Figure 5: Velocity during time

of components $x$ and $y$ of velocity for a point, i.e. `tot_velocity_x.csv` and `tot_velocity_y.csv`, are shown (see tab 6, 7).

| | |
|---|---|
| mean | 0.049947 |
| std | 0.001580 |
| min | 0.000000 |
| 25% | 0.049998 |
| 50% | 0.049998 |
| 75% | 0.049998 |
| max | 0.049999 |

Figure 6: Summary statistics about $U_x$

| | |
|---|---|
| mean | 0.000011 |
| std | 0.000018 |
| min | 0.000000 |
| 25% | 0.000009 |
| 50% | 0.000010 |
| 75% | 0.000010 |
| max | 0.000505 |

Figure 7: Summary statistics about $U_y$

*4.2. Chosen models*

We are considering a supervised learning problem, indeed training data with known target attributes is given. The goal of this project is to predict the components of velocity and pressure, therefore we decide to use a LSTM network and a CNN network to perform a regression. Indeed, a LSTM network is suitable when you are dealing with sequential data or data with temporal relationships, in particular with long-term dependencies as in our case.

In this section, we describe two different forecasting schemes: LSTM for Multiple-Parallel series and CNN, both with vector outputs.

**LSTM for Multiple-Parallel series**

Like other neural networks, LSTMs are able to map input data directly to an output vector that can represent multiple output time steps.

We can predict the next timesteps beyond the end of the dataset by providing as inputs the *n_steps* previous values of the time series. We expect from the model that the shape of the single sample of input data when making the prediction must be [*n_samples*, *n_timesteps*, *n_features*].

In this project, we use an LSTM with two hidden layers composed by 100 neurons and which has ReLU (Rectified Linear Unit) as activation function for the velocities, while tanh for the pressure (indeed, ReLU is inappropriate for its forecast, since this gives as result all NaN values). Finally, there is an output layer with dimension equal to the number of features, i.e. the number of points. To optimize the network, we choose to utilise Adam, an extension to stochastic gradient descent, adopting as the loss function the MSE (Mean Squared Error), which is defined as:

$$MSE = \frac{1}{n}\sum_i (y_i - \hat{y}_i)^2$$

where $n$ is the number of the computed forecasts, $y_i$ are the real values and $\hat{y}_i$ are the predicted values.

**CNN networks**

Convolutional Neural Networks are a type of neural network model which use convolutions in place of general matrix multiplication.

Many research has shown diverse advantages in using CNNs for time series classification over other methods: for instance, they are resistant to noisy data and they are able to extract informative features and informative representation of time series without an explicit order. The model is defined as a *Sequential Keras model*. In our case, we define the model as having one 1D CNN layer, followed by a dropout layer for regularisation, then a pooling layer, even though it is common to define CNN layers in groups of two to give the model a good chance of learning features from the input data. CNNs learn very quickly, so the dropout layer is intended to help slowing down the learning process and hopefully result in a better final model. The pooling layer reduces the learned features to 1/4 their size, consolidating them to only the most essential elements.

After the convolutional and the pooling layers, the learned features are flattened to one long vector and pass through a fully connected layer, before using the output layer to make a prediction. The fully connected layer ideally provides a buffer between the learned features and the output with the intent of interpreting the learned features before making a prediction.

For this model, we will use a standard configuration of 64 parallel feature maps and a kernel size of 2. The fea-

ture maps are the number of times the input is processed or interpreted, whereas the kernel size is the number of input time steps considered as the input sequence is read or processed onto the feature maps.
The efficient Adam version of stochastic gradient descent will be used to optimize the network, and the categorical cross-entropy loss function will be used, since we are learning a multi-class classification problem. The model is fit for a fixed number of epochs, in this case 3000, and a batch size of 32 samples will be used, where 32 windows of data will be exposed to the model before its weights are updated. Finally, its evaluation is computed on the test set.

## 5. Mathematical Description of the proposed models

Let's now describe mathematically the chosen models, pointing out that LSTMs are a special kind of RNNs.

### *5.1. RNNs*

We choose to use RNNs because in RNNs we feed the hidden state from the previous time step back into the network at the next time step, so regarding our problem it is useful because our data represents the flow during the time, therefore we can learn dependencies.

The following figure shows a graphic representation of a Recurrent Neural Network:

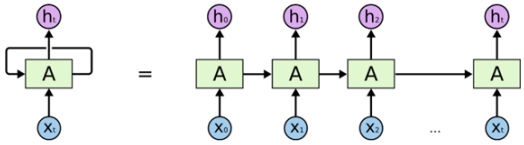


Figure 8: Graphical representation of a RNN (image from [4])

In the last few years, there have been incredible successes applying RNNs to a variety of problems. Fluid dynamics also proved to be an interesting field in which to apply this model. When it comes to analyzing the trend of a fluid, it is certainly interesting to consider the dynamics in the previous time steps to predict the future dynamics.

However Recurrent Neural network can present several issues, for example the vanishing gradient problem or the exploding gradient problem. Indeed, a neural network uses the algorithm called *BackPropagation* to update the weights of the network. So the algorithm first calculates the gradients from the error using the chain rule, then it updates the weights using Gradient descent. The BackPropagation starts from the output layer and it propagates to all the way back to input layer, when we work with deep neural networks this can lead to some issues.

In RNN's, we use time steps and current time step value depends on the previous time step so it is needed to go all the way back to make an update.

### *5.2. LSTM*

**Long Short Term Memory networks** – usually also called **"LSTMs"** – are a type of RNN capable of learning long-term dependencies. In particular, it is decided to use the LSTM model because it can process not only single data points but also entire sequences of data. Therefore, after we split data into sequences, we then use LSTM to predict entire sequences of data.

The model is used to predict the pressure and the velocities components.

It is implemented through the use of the following python libraries:

- *Sequential from keras.models;*
- *LSTM from keras.layers;*

- *Dense from keras.layers;*

LSTMs are explicitly designed to avoid the long-term dependencies problem that may be present in RNNs.

LSTMs have a chain like structure, there are four neural network layer, interacting with each other.

A common LSTM is composed of:

- Cell
- Input Gate
- Output Gate
- Forget Gate

The cell remembers values over an arbitrary time interval, and the three gates regulate the flow of information.

So, in particular, the memory cell present a structure like:

WRITE X,M

READ M,Y

FORGET M

Inputs to the LSTM cell at any step are current input, previous hidden state, and previous memory state. Outputs from the LSTM cell are the current hidden state and the current memory state.

The following image presents a diagram for a LSTM cell at a $T$ time step:

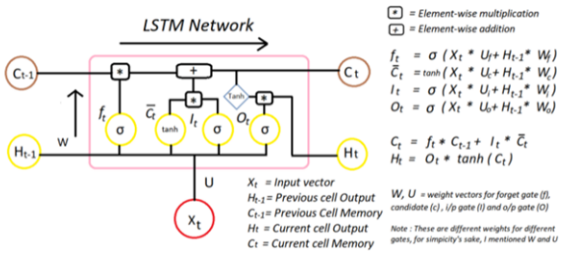


Figure 9: Diagram for a LSTM cell (image from [6])

In particular the forget gate, the candidate gate, the input gate and ouput gate are single layered neural networks with the Sigmoid activation function except candidate layer (it takes tanh as activation function).

These gates first take input vector and previous hidden state, then concatenate them and apply activation function. Finally these gate produce vectors - between 0 and 1 for Sigmoid, -1 to 1 for tanh -, so we get four vectors for every time step.

Then, LSTM cell takes the previous memory state $C_{t-1}$ and does element wise multiplication with forget gate.

If forget gate value is 0, then the previous memory state is completely forgotten, while, if it is 1, the previous memory state is passed to the cell.

Now, we compute a new memory state from the input state and the layer $C$ with current memory state $C_t$.

The output is based on our cell state $C_t$. Therefor, it is applied tanh to $C_t$ and, then, we do element wise multiplication with the output gate O.

For our problem, after splitting our data into **Training** and **Test set** (80% - 20%) choosing chronologically we convert our data into input and output, after choosing the time steps, we define the model and fit the model in the Training set.

Therefore we check if there are some *NaN* or infinite items in the dataset, and finally we proceed with the prediction. In the end we save the data.

We complete the prediction for all components under analysis.

### *5.3. CNN*

Convolutional networks are inspired by biological processes in case of connectivity patterns of neurons.

The convolution kernels always have the same width as the time series, on the other hand, their length can be varied. This means that, the kernel moves in one direction from the beginning of a time series towards its end, performing convolution.

The elements of the kernel are multiplied by the corresponding elements of the time series that they cover at a given point, the results of the multiplication are added together and a nonlinear activation function is applied to the value. The output becomes an element of a new "filtered" time series, and then the kernel moves forward along the time series to produce the next value. The number of new "filtered" time series is the same as the number of convolution kernels.

Different properties of the initial time series get captured in each of the new filtered series, depending on the length of the kernel. The next step is to apply global max-pooling to each of the filtered time series vectors: the largest value is taken from each vector. CNN uses max pooling to replace output with a max summary to reduce data size and processing time. This allows us to determine features that produce the highest impact and reduces the risk of overfitting. A new vector is formed from these values, and this vector of maximums is the final feature vector that can be used as an input to a regular fully connected layer.

So after each convolutional and max pooling operation, we can apply Rectified Linear Unit (ReLU). The ReLU function mimics our neuron activations to introduce nonlinearity for values $x > 0$ and returns 0 if it does not meet the condition. This method is used to solve diminishing gradients. Weights that are very small will remain as 0 after the ReLU activation function.

In particular, in the end, we will serve the convolutional and max pooling feature map outputs with Fully Connected Layer (FCL). We flatten the feature outputs to column vector and feed-forward it to FCL, then we use the softmax activation function where we wrap our features with softmax activation function which assign decimal probabilities for each possible label which add up to 1.0. Every node in the previous layer is connected to the last layer and represents which distinct label to output.

## 6. Performance and model evaluation

To evaluate the forecast accuracy, various metrics are utilised, following what is described in [11]:

- MAE (Mean Absolute Error)
  Given $e_t = y_t - \hat{y}_t$, where $\hat{y}_t$ is the predicted value and $y_t$ is the true value, MAE is equal to:

  $$\text{MAE} = \text{mean}(|e_t|)$$

  It is really spread, since it's easy to compute and interpret.

- RMSE (Root Mean Squared Error)
  Analogous to MAE, it's defined as:

  $$\text{RMSE} = \sqrt{\text{mean}(e_t^2)}$$

  It is popular too, although it is not as easy to understand as MAE.

- MAPE (Mean Absolute Percentage Error)
  Considering the percentage error $p_t = 100e_t/y_t$, MAPE is analogous to MAE and equal to:

  $$\text{MAPE} = \text{mean}(|p_t|)$$

Note that this metric puts heavier penalty on negative errors than on positive errors.

- sMAPE (symmetric MAPE)
  The asymmetry of MAE leads to define this metric:

$$\text{sMAPE} = \text{mean}\left(\frac{200|y_t - \hat{y}_t|}{|y_t| + |\hat{y}_t|}\right)$$

  Despite its name, sMAPE can assume values up to 200, so it's not a percentage.

The first two metrics are scaled-dependent, so they shouldn't be used to compare series with different units. In this case, this isn't a problem. Instead, MAPE and sMAPE have the advantage to be unit free, but at the same time they are unstable with $y_t$ close to zero: for this reason, they shouldn't be appropriate for this problem.

## 7. Results obtained and their interpretation

Since we have 11160 parallel series, the accuracy is evaluated for each considered point. To sum up the obtained performance evaluations, we decided to take the mean of all these 11160 accuracy values for each model and the results are shown in the tables 10, 11.

| | MAE | RMSE | MAPE | sMAPE |
|---|---|---|---|---|
| pressure LSTM | 5,03E-05 | 5,93E-05 | inf | 63,1966 |
| Ux LSTM | 4,86E-04 | 6,27E-04 | inf | 13,2037 |
| Uy LSTM | 4,40E-04 | 5,58E-04 | inf | 49,1793 |
| Uz LSTM | 2,48E-22 | 2,09E-22 | inf | 154,5083 |

Figure 10: Accuracy LSTM

| | MAE | RMSE | MAPE | sMAPE |
|---|---|---|---|---|
| pressure CNN | 5,00E-05 | 5,85E-05 | inf | 70,7409 |
| Ux CNN | 4,47E-04 | 5,77E-04 | inf | 13,1837 |
| Uy CNN | 2,47E-04 | 3,13E-04 | inf | 30,0731 |
| Uz CNN | 2,09E-22 | 2,47E-22 | inf | 154,5083 |

Figure 11: Accuracy CNN

Both LSTM and CNN models request considerable computational costs, but, observing the obtained values of MAE and RMSE, they seem to work well. Instead, we shouldn't rely on values of MAPE and sMAPE: in this case, MAPE is always infinite because there is at least one $y_t = 0$ and, analogously, sMAPE is unstable, because with any $y_t$ that tends to zero, it booms up to its upper-limit (200).
Comparing LSTM and CNN, they actually seem to obtain very similar results; the latter provides values of accuracy which are just a little better.

Now, let's compare the pressure predictions done by LSTM and CNN. First of all, we consider the pressure of every point measured at a particular second and we realize a scatterplot with real values on the ax $x$ and the predicted pressures on the ax $y$: more the dots are near to the quadrant bisector, better is the forecast.

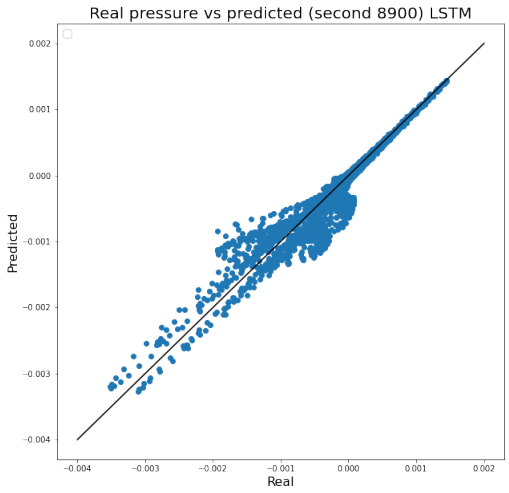


Figure 12: Comparison between real and predicted pressures for LSTM

Both the figures 12 and 13 show good results, but CNN seems to be better (as we expected by the accuracy values). Moreover, we can notice that with the highest values of pressure, predictions look more precise. Any-

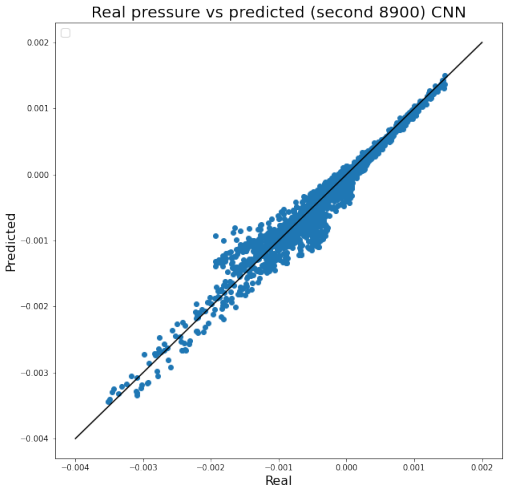


Figure 13: Comparison between real and predicted pressures for CNN

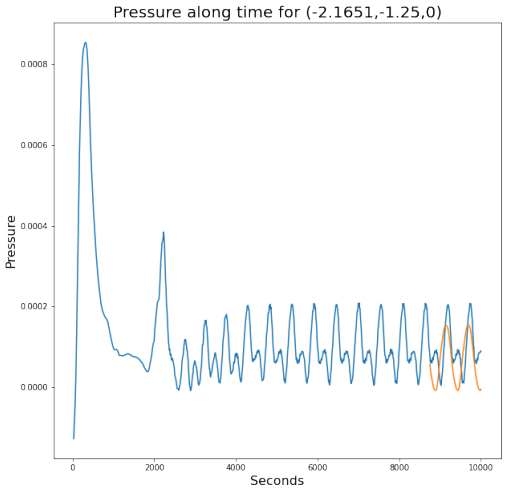


Figure 15: Real and predicted pressure of a point with CNN

way, this observation can also be trivially explained by the fact that with lower values there are more points to predict.

Choosing one point, a temporal series with real and predicted values of pressure evolving during the time is plotted for both LSTM and CNN. Surely, these series can be drawn for more points, but we report only one example for each model (see fig. 14, 15).

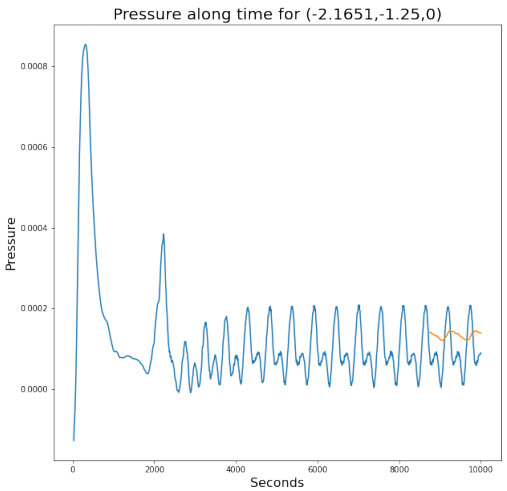


Figure 14: Real and predicted pressure of a point with LSTM

Here, it's more evident that CNN works better than LSTM. It's also clear that we shouldn't take for granted the fact that these models have satisfying results: this actually depends on the precision we aim for our goals. It's in fact possible to see clearly the difference between the real and the predicted temporal series.

Let's now focus on the velocity: real vectors $U = (U_x, U_Y)$ are compared to the predicted one for both LSTM and CNN. In particular, the scatterplots 16, 17 represent every point velocities at second 9990, where the dots correspond to the extremities of the velocity vectors. The results appear to be really good and the utilised models seem to give very similar predictions.

Now, we consider only a few points to compare their real and predicted velocity vectors: these graphics show better the differences between the velocities than the scatterplot, since their modules vary a lot with the change of points, so, considering all the points at once causes a messy visualization. Anyway, the previous graphics give an overall idea of the forecasts; still, to analyse more in details the predictions, it's necessary to consider only a restricted set of points.

In the figures 18, 19, we consider only five points in the space ((-24.500000,-9.333300),

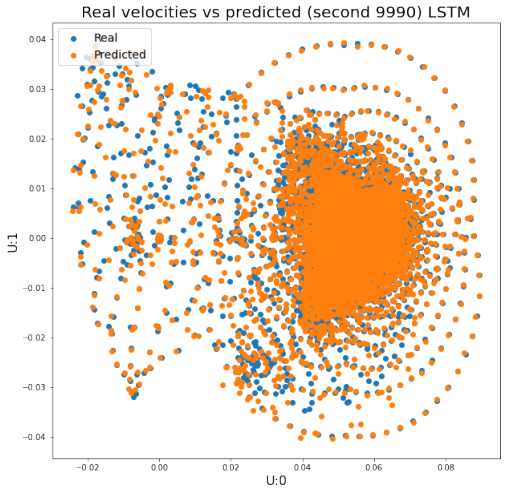


Figure 16: Comparison between real and predicted velocities for LSTM

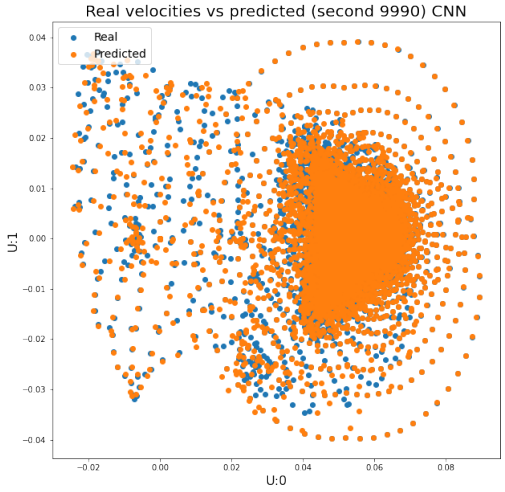


Figure 17: Comparison between real and predicted velocities for CNN

(0.050418,-0.000112), (0.050000,-0.000006), (0.054242,0.000324), (0.050129,-0.003241)): it's possible to see that LSTM provides a good prediction, but obviously not perfect, so it's really important, if anyone wants to use this algorithms, to establish which level of precision is aimed.

Even in this case, for the same points CNN gives better results.

Obviously, it's possible to print graphics for more

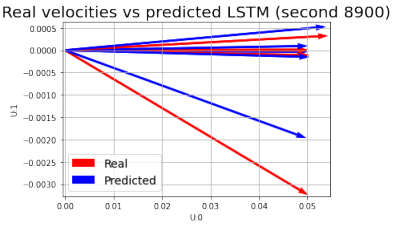


Figure 18: Comparison between real and predicted velocities for LSTM

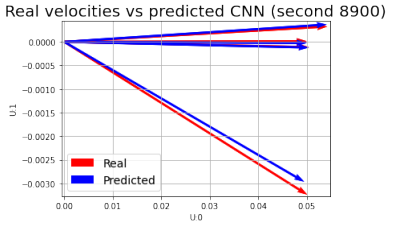


Figure 19: Comparison between real and predicted velocities for CNN

points, but it's not recommended to use it with many points, otherwise it will be very chaotic and hard understandable (anyway, the code is attached to the project).

In the end, we can say that we obtained really good results: that warns us, because too positive outcomes can be a sign of model overfitting. Surely, further controls of the models correctness are needed.

## 8. Conclusion

The usage of Machine Learning and Deep Learning models in the field of fluid dynamics can influence significantly researches quality and their computational costs. This project focuses on the problem of the flow around a circular cylinder, searching for an alternative to the computationally demanding numerical solutions. In particular, we applied two Neural Networks model: LSTM and CNN.

The obtained results seem to be very good and encourage further researches for this scope. In particular, we didn't consider the relationship between pressure and

velocities and between the velocity components: certainly, it would be interesting to implement a model that take them into account.

Moreover, we restricted our focus only on two models, but analysing the results with algorithms of different kind could be interesting. Anyway, the complexity of the problem doesn't permit to apply too simple models. At the end of the performance evaluation, we conclude that CNN is the winning choice: as can be seen particularly from the graphics, it provides better predictions. Moreover, the computational costs are analogous between the two methods. A stimulating evolution of this project can be the combination of the two utilised models: the same forecast can be realised even by a **CNN LSTM** Network (see [9]).

What concerns us most about these results is the risk that they could be born from overfitted models. For this reason, it would be appropriate to apply these models even to new data to verify their goodness.

## References


[1] Sangseung Lee, Donghyun You, Data-driven prediction of unsteady flow fields over a circular cylinder using deep learning in *Journal of Fluid Mechanics*, 2019, pp.217-254

[2] Xiaowei et al. and Peng and Wen-Li and Hui, Prediction model of velocity field around circular cylinder over various Reynolds numbers by fusion convolutional neural networks based on pressure on the cylinder in *Physics of Fluids*, 2018, volume 30

[3] Tatan, Understanding CNN (Convolutional Neural Network), https://towardsdatascience.com/understanding-cnn-convolutional-neural-network-69fd626ee7d4, DEC 2019

[4] Oinkina, Understanding LSTM Networks, https://colah.github.io/posts/2015-08-Understanding-LSTMs/, FEB 2022

[5] Santhoopa Jayawardhana, Sequence Models & Recurrent Neural Networks (RNNs), https://towardsdatascience.com/sequence-models-and-recurrent-neural-networks-rnns-62cadeb4f1e1, JULY 2020

[6] Madhu Sanjeevi, Chapter 10.1: DeepNLP — LSTM (Long Short Term Memory) Networks with Math., https://medium.com/deep-math-machine-learning-ai/chapter-10-1-deepnlp-lstm-long-short-term-memory-networks-with-math-21477f8e4235, FEB 2022

[7] Anne Bonner, Convolutional Neural Networks and Image Classification, https://towardsdatascience.com/wtf-is-image-classification-8e78a8235acb, FEB 2019

[8] Jason Brownlee, How to Develop LSTM Models for Time Series Forecasting, https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/, FEB 2022

[9] Jason Brownlee, CNN Long Short-Term Memory Networks, https://machinelearningmastery.com/cnn-long-short-term-memory-networks/, FEB 2022

[10] Margarita Granat, How to Use Convolutional Neural Networks for Time Series Classification, https://towardsdatascience.com/how-to-use-convolutional-neural-networks-for-time-series-classification-56b1b0a07a57, OCT 2019

[11] otexts.com, Evaluating forecast accuracy, https://otexts.com/fpp2/accuracy.html, FEB 2022