Imaging time series to improve NN forecasting

Alper Karaaslan, Francesco Forcher, Simeone de Fremond, Stefano d'Apolito

***Abstract*— In this project we have explored the use of imaging time series to enhance forecasting results with Neural Networks. The approach has revealed itself to be extremely promising as, both in combination with an LSTM architecture and without, it has out-performed the pure LSTM architecture by a solid margin within our test datasets.**

I. Introduction

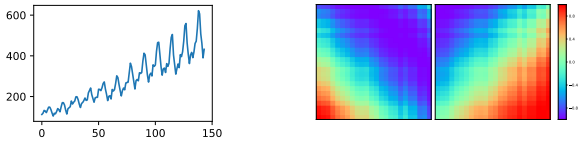
The classical approach to time series forecasting is the use of linear statistical methods such as ARIMA models. Newer approaches, however, such as neural network (NN) based methods have become serious candidates for dealing with non linear time-series, winning forecasting competitions such as NN3, NN5, ESTSP and M4 [1] [2]. The task of forecasting is particularly challenging for NNs as classical architectures do not have a notion of time axis. Additionally a forecasting model should manage to capture long term and short term patterns. For example, forecasting models for occupancy of freeways should manage to recognize morning peaks vs. evening peaks, workdays vs week-ends; the ones for solar energy production of a farm should recognize long patterns (day-night cycle) and short ones (movements of clouds) [3]. Usually LSTM and GRU architectures are used for these tasks. A second possible approach is to "visually" learn temporal correlations and patterns by transforming the time-series into images with an appropriate mapping $M : R^n \rightarrow R^{n \times n}$ that captures and highlights the information that identifies the time-series. We can characterize a time-series with the temporal correlations, i.e. the covariances between a value at time t and a value at time $t - k$ for any t and k inside the temporal discrete domain of the time-series. Naturally, we can express those covariances in a square matrix, and those can be seen as images, hence the duality between time-series and images. An important step in the direction of imaging time series for NNs was made by [4] that proposed Gramian Angular Fields (encoding time-series covariances) and Markov Transition Fields (dynamical transition statistics) for time-series classification and imputing. The method works well for time-series classification because imaging enables us to use the powerful techniques developed in the field of computer vision.

In this project, we went further and tried to find out if, coupled with a windows approach, the transformations proposed by [4] could also be used effectively for time-series forecasting. We have tested the performance of this approach (with two different NN architectures) against the multiplicative Holt-Winter model and a classical LSTM architecture.

Furthermore, some outputs of convolutional layers have been included to show how the filters reacted when presented with an image from a transformation. The exploration of how the network used assignments of "importance values" to specific pixels of an image for forecasting using activation maps has also been shown.

II. Models and Methods

We will describe, here, the ideas that led to our forecasting model as well as give a concise overview of the Gramian Angular Fields (GAFs), the Markov Transition Fields (MTF) [4] and the Holt-Winter method. Furthermore, we will describe the neural network architectures that we have coupled with our approach, and the analysis technique we used to visualize the action of convolutional layers.



(a) Airplane traffic over 144 months

(b) GAFs Transformations

Fig. 1: Transformation from time-series to images

A. GAFs and MTF

Given a time-series $X = \{x_1, x_2, \ldots, x_n\}$, the values can be rescaled in the interval [-1,1], and for each of them, $\phi_i = \arccos(x_i)$ can be computed. The element (i, j) of the matrix relative to the Gramian Summation Angular Field then is $GASF_{ij} = \cos(\phi_i + \phi_j)$, whereas for the Gramian Difference Angular Field is $GADF_{ij} = \sin(\phi_i - \phi_j)$. The transformation preserves time dependencies as time increases, going from the top-left corner of the matrices to the bottom-right. Moreover, the elements with indices $(i, j)||i-j| = k$ capture the temporal correlation relative to the time interval k by superposition/difference of the time-series' values.

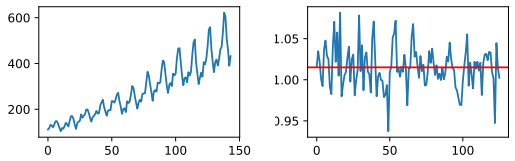
The element (i, j) of the matrix relative to the Markov Transition Field $MTF_{i,j}$ is related to the probability of transition from the value held by x_i to the one held by x_j in a temporal interval $k = |i-j|$. To see the exact way in which the MTF matrix is computed we refer to [4].

B. Improving robustness of the model against scale variance

Usually pure NN models have problems with time-series' scale variance [3] so they are often combined with classical models. For example in [3] the prediction is decomposed into a linear part (focusing on local scaling issues) using an Auto Regressive model and a non linear part (focusing on recurring patterns and temporal correlations) using an LSTM architecture. [2] integrates, in a unique model, a Recurrent Neural Network with the Holt-Winter (HW) model. Since HW is a fairly simple model, that manages to capture the essential skeleton of a time-series' seasonality and scale, that smooths the noise out, and that has already been adapted by [2] with a windows approach, we choose to implement our forecasting NN architecture on top of it.

C. Multiplicative Holt-Winter model

The multiplicative Holt-Winter model (HW) contains a forecasting equation (predicting h steps ahead) and 3 exponential smoothing equations for the 3 hidden states. l is the "scale" of the time series, b the local linear trend, and s is the multiplicative seasonality coefficients (with season of length m). α, β, γ are the



(a) Original past values **(b)** Vector X_t

Fig. 2: Normalization of the past values of the time-series through s and l, airplane traffic

smoothing factors. The model then is:

$$\hat{y}_{t+h|t} = (\ell_t + hb_t)s_{t+h-m(k+1)} \tag{1a}$$

$$\ell_t = \alpha\frac{y_t}{s_{t-m}} + (1-\alpha)(\ell_{t-1} + b_{t-1}) \tag{1b}$$

$$b_t = \beta^*(\ell_t - \ell_{t-1}) + (1-\beta^*)b_{t-1} \tag{1c}$$

$$s_t = \gamma\frac{y_t}{(\ell_{t-1} + b_{t-1})} + (1-\gamma)s_{t-m} \tag{1d}$$

D. Our model

In our model we are going to use a window approach. Indeed, we will train a NN to capture the time correlation between x_{t+h} and $x_{t-i}, \forall i \in \{0, 1, ..windowsize - 1\}$. Furthermore, imaging time-series allows the NN to directly access the temporal correlations between the values inside the window. Having said that, *windowsize* is an important hyper-parameter, because the NN will not learn potential temporal pattern with values further in the past than *windowsize*. Nevertheless, these old values are often obsolete, and they usually do not convey any useful information in real applications.

Since the NN essentially does the same work as the hidden state b (except that it works in a multiplicative way and captures also non linearities), we choose to discard the smoothing equation (1c) of b. [4]. Hence our model becomes:

$$l_t = (y_t/s_{t-m})\alpha + (1-\alpha)l_{t-1} \tag{2a}$$

$$s_t = (y_t/l_t)\gamma + (1-\gamma)s_{t-m} \tag{2b}$$

$$\hat{y}_{t+h} = \text{NN}(\text{img}(X_t))l_t s_{t+h-m} \tag{2c}$$

where $X_t \in R^n$ is a vector whose components $x_i = \frac{y_i}{l_i s_i}$ are the normalized values of the current window, $\text{img}(X_t)$ is the image representation of X_t and $\text{NN}(\text{img}(X_t))$ is the regressed value of the NN based on it. $\text{img}(X_t)$ will be the stack of the two Gramian Angular Fields (GADF and GASF) and the Markov Transition Field (MTF). The 3 values for a single index (i, j) can be thought of as the RGB colours of the image.

The effect of the normalization through the state l and s of the model i.e. the transformation of the old values of the series into the normalized one taken by X_t can be seen in figure 2. After the normalization, the NN only has to focus on the non linear temporal correlations, as the scale and the seasonality can be easily estimated by HW. A comparison can be imagined between our model and a hypothetical pure HW where the hidden state b, instead of being used in an additive mode, is used in a multiplicative manner (similarly to the way the NN is used in our approach). Then the 2 models would differ in that b would always yield a value that is a weighted average of the values of X_t (about 1.015 in figure 2), while the NN would yield a learned factor mapping X_t (through the images) to $\frac{y_{t+h}}{l_t * s_{t+h-m}}$.

E. LSTM

As a baseline to compare against our approach we have chosen a forecasting method based only on a Long-Short-Term-Memory architecture. This popular architecture was designed specifically to capture long and short relations/patterns in a sequence of elements (e.g. a time-series).

F. Convolutional-MLP

Deep Networks are extremely successful in Computer Vision tasks, such as image recognition. In this framework, the most common architectures are made up of multi layer perceptrons (MLP) on top of convolutional layers. Naturally we choose to use, as our first model (involving temporal correlation regression through images), this type of architecture. We expect the convolutional layers (combined with pooling) to extract the most useful information while reducing the dimensions of the input, and the MLP to regress a value based on them. Full details about our network architecture can be found in the Appendix. 12.

G. Convolutional-LSTM

As discussed before, one possible weak point of our model is that it could lose really long time patterns recognition, i.e. the ones with length longer than the size of the window. As a possible solution, we have decided to wrap the convolutional network in a time distributed layer to obtain our feature vectors in a sequence which then can be fed into an LSTM network (3). This layer has the ability to memorize possible signals coming from windows in the past thanks to its recurrent cell structure. We choose the LSTM layers to be bidirectional because the generative law underlying the time-series could be inferred from both time directions. The LSTM then goes through a batch normalization layer before being passed to a MLP network to get our predictions of the output window. Full details about our network architecture can be found in the Appendix 13.

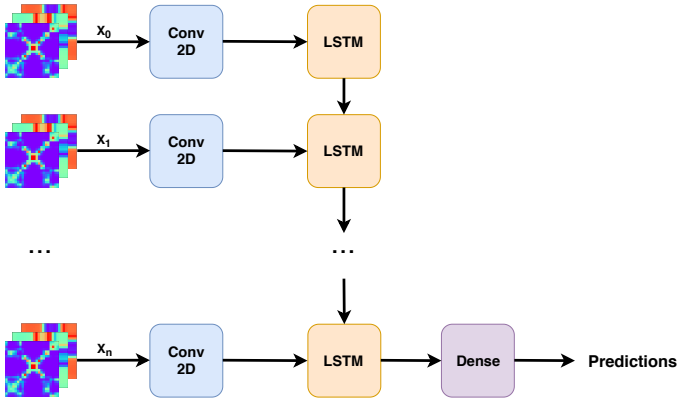


Fig. 3: The ConvLSTM network takes as input a sequence n of combinations of gadf, gasf and mtf images and outputs predictions

H. Interpretability of a convolutional networks

There are several ways to visualize what the network learns. One way would be to monitor the output of an activation layer given an input image and showing how the convolutional filters acted upon it. Another way would be to give the system a noise image as input and check the reaction of a filter at a chosen layer. This allows us to see what activates a kernel. One of the most exciting visual explanations are the so called class activation maps (CAM) [5]. With the CAM, a heatmap can be overlayed on top of the

input image, which shows how "important" several pixels are with regards to their contribution to the final decision. This could be used as a sanity check to see if some random patterns are considered important or rather specific regions of an image.

III. Results

We will now describe the types of time-series that we have used to test our model, and we will then report the relative experimental results.

A. Test data-set

Different models have been used to generate the data which are detailed below.

The first type of time-series follows the law:

$$y_t = cost + trend(t) * s_m(t) + ma_t^{\sigma} \tag{3}$$

where $cost$ is a constant, $trend(t)$ is a function representing the trend of the series, $s_m(t)$ is a function with period equal to m, representing the seasonal factor, and $ma_\sigma(t)$ represents the stochastic part of the series, in particular:

$$ma_t^{\sigma} = \sum_{i=1}^{t_{\max}} \alpha_i * ma_{t-i}^{\sigma} + \eta_t \tag{4}$$

$$\eta_t \sim \mathcal{N}(0, \sigma^2) \tag{5}$$

This is typically a good model for many real-world time-series, such as the airplane traffic time-series in Fig. 1a. For the experiment we have chosen $trend(t)$ and $s(t)$ to be a combination of sine and cosine functions multiplied with different factors, so that we are able to test the ability of the predictors to learn patterns with different scale. We will refer to this dataset as the "Noisy Dataset" or "Noisy Function".

The second type of time-series is a one-dimensional projection of the numerical solution of the Lorenz attractor ODEs (6), obtained using 4th order Runge-Kutta: the time-series corresponds to the $X-$component of the state. For appropriate parameters (σ, ρ, β), this is modelling a system with a chaotic evolution, in which arbitrarily close states can diverge exponentially fast over time. The system is nonetheless deterministic, depending only on a tridimensional state, which is only partially observed.

$$\begin{cases} \dot{x} &= \sigma(y - x) \\ \dot{y} &= x(\rho - z) - y \\ \dot{z} &= xy - \beta z \end{cases} \tag{6}$$

B. Experimental set-up and result

We tested a number of different methods to perform our predictions: Conv-MLP, Conv-MLP combined with Holt-Winter (Conv-MLP-HW), Conv-LSTM, Conv-LSTM with Holt-Winter (Conv-LSTM-HW), pure Holt-Winter and pure LSTM. For all our experiments we have trained on 200'000 data points and tested on 10'000 data points. In table I we reported the mean squared error (MSE) values for respectively 12 and 10 time steps ahead predictions, on Noisy and Lorenz data-set, based on a window made of the last 40 values of the series. Regarding Conv-LSTM, the LSTM layer has a sequence composed of only the 3 last images (for a total of 42 different past values received), so that the comparison with other methods is still fair in the case long term patterns play an important role. Figure 4 and figure 5 visualize the predictions obtained by the best performing method in both data-sets

	Noisy Function	Lorenz Attractor
HW	647.5	85.1
LSTM	986.2	39.3
Conv-LSTM-HW	**83.4**	13.9
Conv-LSTM	135.6	**11.2**
Conv-MLP-HW	615.7	21.0
Conv-MLP	3134.1	68.4

TABLE I: MSEs of all networks tested on both datasets.

Note that for the noisy data set, the parameter of the series were chosen so that $\text{Var}(ma_{t+12}|ma_t) = 22.8$. Hence this would be the expected MSE of a forecasting method knowing exactly the underlying generating model.

C. Training time

With regards to the same experiments, we report also the mean time of training per epoch (averaged over different runs and datasets but using the same amount of training data points) so that we can also establish a performance comparison:

Conv-MLP	Conv-LSTM	LSTM
55s	305s	800s

TABLE II: Average time of an epoch on 6 core CPU and NVidia GTX 1070 Max-Q GPU

We can notice that networks containing a recurrent layer need more computational resources. Note also, that Conv-LSTM uses a significantly higher amount of space than Conv-MLP, reaching 25Gb of RAM usage during the training, which could likely be mitigated by using a generator function to avoid having to store a tensor of dimensions (number of samples, sequence length, image size, image size, 1).

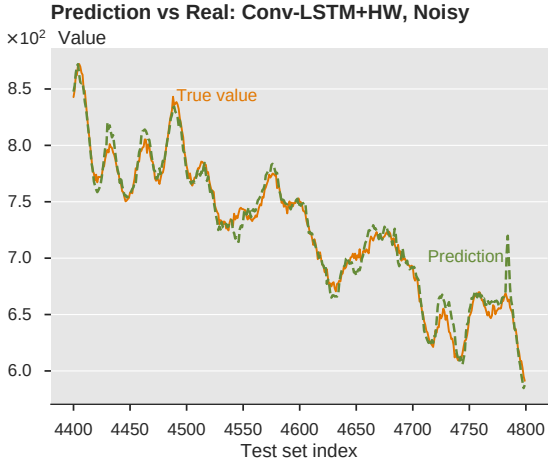


Fig. 4: Predictions 12 time steps ahead by Conv-LSTM-HW for the noisy function

D. Visualization and CAM

To visualize activation maps and convoluted images as discussed in subsection II-H , the Gramian angular field transformed images from the Lorenz attractor dataset were fed into the Conv-MLP architecture: the inputs are the two images on the left side of figure

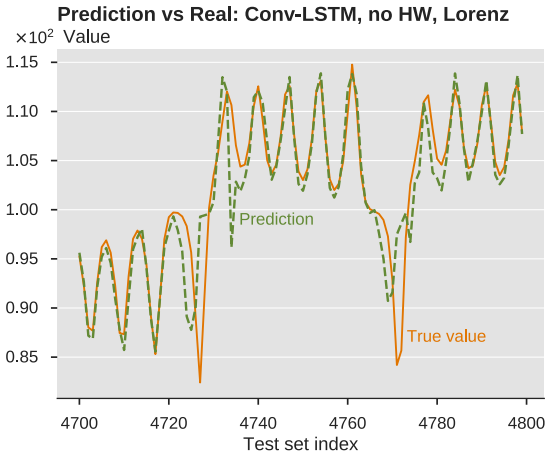


Fig. 5: Predictions 10 time steps ahead by Conv-LSTM for the Lorenz attractor dataset

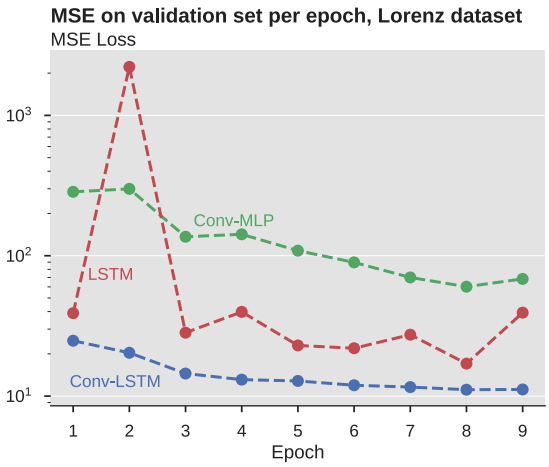


Fig. 6: MSEs over epoch of the tested methods

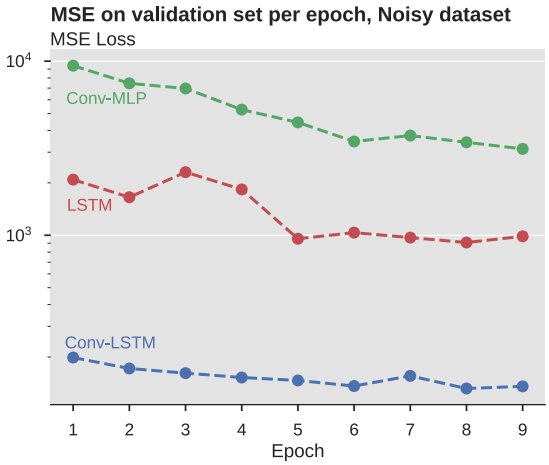


Fig. 7: MSEs over epoch of the tested methods

8 and the activation map is visible on the right side. Figures 9 and 10 show the outputs of both the first and last convolutional layers of the Conv-MLP architecture given the input images.

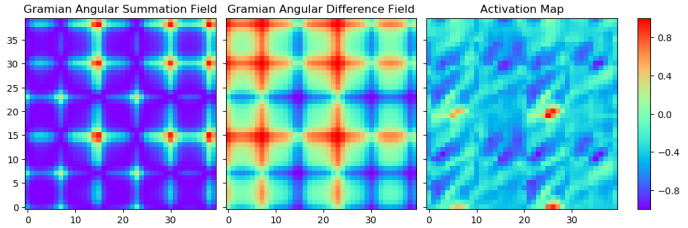


Fig. 8: The first two images on the left are the input images for the activation map on the rightmost image

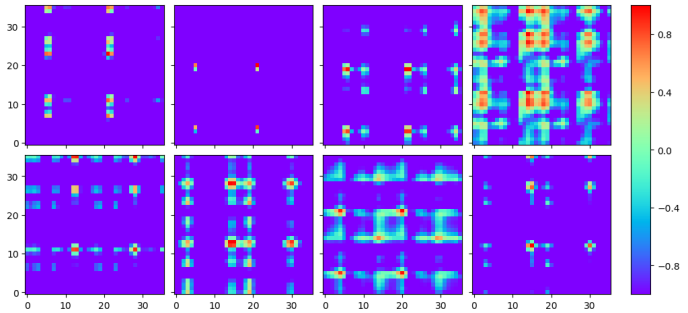


Fig. 9: The outputs of the first convolutional layer

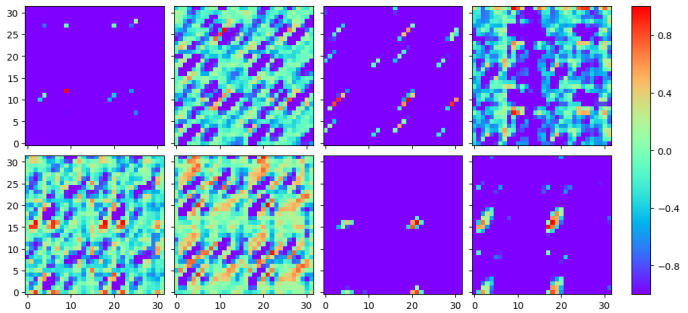


Fig. 10: The outputs of the last convolutional layer

IV. Discussion

A. Performance of the different forecasting methods

Across both datasets, we noticed that the winners are the Conv-LSTM models with and without Holt-Winter. It is interesting to notice that in our experiments the LSTM layer was using only 3 overlapping images (each following image is the one of before shifted by one pixel toward NE) which barely brings more information than just one window, and yet the addition of this layer brought a decisive improvement. The interpretation of this result is mysterious. Perhaps the fact that these neural networks have the opportunity to learn different weights from the same images each time slightly shifted could have helped, or maybe the LSTM really manages to learn important features for the prediction of $10/12$ steps in the future only from the sequence of the last 3 images.

Having a look at the other results, it can also be observed that the Conv-MLP on top of HW performs well even without the help of a single memory unit. Indeed, it manages to out-perform LSTM on

both datasets. This success can not only be attributed to the Holt Winter component of the method as Conv-MLP-HW does better than pure HW with a solid margin in the Lorenz data-set. Looking at the prediction graph 11 of Conv-MLP on noisy, we can see that its low accuracy does not come from the fact that it did not manage to capture the temporal patterns of the series, but rather from the fact that it is constantly predicting values in the wrong scale, i.e. the shape of the predicted graph resembles the original one but is shifted too high or too low. This demonstrates the value of the contribution of HW when combined with a NN. Similarly, but on a much smaller scale, the LSTM suffers from the same issue. The Lorenz attractor dataset, on the other hand, does not offer much challenges from a scale point of view, hence Conv-MLP's ability to perform decently. Conv-LSTM, however, looks like it does not need the help of the HW infrastructure as it is already a really strong forecaster, but maybe when faced with a dataset that has more challenging scale variation, the HW could prove itself to be a beneficial addition to the network.

Comparing the results against the other baseline, the pure Holt-Winter method, NNs using imaged time-series perform better in both datasets, even if it must be said that Holt-Winter is not a suitable method to face the Lorenz time-series. We have noticed, however, from the other test with the noisy dataset, that the noisier the dataset, the smaller the gap between NNs and HW.

Finally, we did also a little experiment based on the daily value of the Dow Jones financial index in the last 3 decades. We trained the data over the first 4000 samples and tested the results on the next 4000. This series is very hard to predict beyond a simple random walk with drift model, and the number of samples used was likely too low for neural network methods. What emerged though was that the methods using HW were more robust, in the sense that they still managed to achieve reasonable results (close to the one obtained with naive forecasting, i.e. using the last value of the series as prediction). We report the MSE obtained in the following table:

Conv-MLP	Conv-LSTM	Conv-MLP-HW	Conv-LSTM-HW	LSTM
1.458	2.401	0.013	0.017	0.020

TABLE III: MSE$\times 10^8$ of models on Dow Jones, with horizon of prediction h=6 days

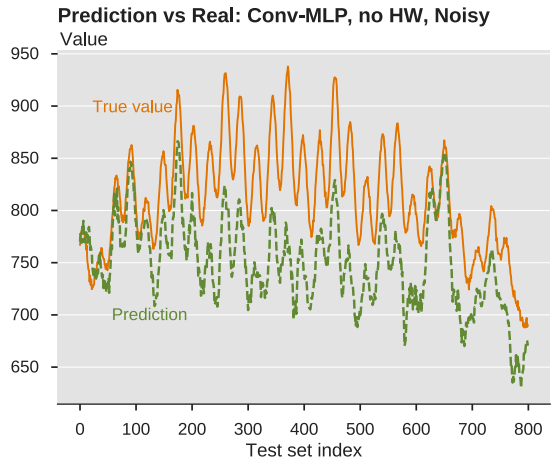


Fig. 11: Predictions 12 time steps ahead by Conv-MLP for the noisy function

B. *Visualizations*

When comparing the outputs of the first and the last convolutional layers,(figure 9,10) it can be seen that they lose the typical structure of the original transformed image. In both cases, they show some well defined structure that seem far from being the result of chance. The heatmap on 8 reveals where Conv-MLP focused its attention given a window coming from the Lorenz dataset. In this case, the hot points are the correlations between the oldest and the central values of the window with the ones in positions $5-7$ and $25-27$

V. Conclusion

In this project we have examined the possibility of using the method of imaging time-series to improve forecasting as opposed to methods based purely on memory units. Additionally, we have evaluated whether it is actually useful to combine the neural networks with classical methods, in particular with Holt-Winter. The results show that the proposed methods yield very good predictions even against particularly challenging time-series, such as the one derived from the Lorenz attractor, beating methods based on a basic LSTM. It also took vastly less time to train the network, albeit with increased memory requirements compared to standard LSTM. Incorporating Holt-Winter led to more robust results (especially with noisier and multi-scale series) at the cost of an increased complexity of the method.

Our experiments suggest that an interesting step further would be to further investigate the combination of the images approach with an LSTM architecture. More specifically, one could feed the LSTM layer with images derived from consecutive windows (as this appeared to sharply boost performance) or from non overlapping ones. The rationale behind this second choice is to achieve an architecture that can capture really long time dependencies but, at the same time, avoiding the explosion of the images' dimensionality (which depends on the square of the window size nw) and reducing the number of memory units by a factor of nw.

APPENDIX

ARCHITECTURES OF THE NEURAL NETWORKS

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 36, 36, 8)	408
conv2d_1 (Conv2D)	(None, 32, 32, 8)	1608
batch_normalization (BatchNo	(None, 32, 32, 8)	32
max_pooling2d (MaxPooling2D)	(None, 16, 16, 8)	0
dropout (Dropout)	(None, 16, 16, 8)	0
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 48)	98352
dense_1 (Dense)	(None, 32)	1568
dense_2 (Dense)	(None, 1)	33

Total params: 102,001
Trainable params: 101,985
Non-trainable params: 16

Fig. 12: Architecture of the Conv-MLP network

Layer (type)	Output Shape	Param #
input_3 (InputLayer)	[(None, 3, 40, 40, 3)]	0
time_distributed_1 (TimeDist	(None, 3, 4096)	26780
batch_normalization_13 (Batc	(None, 3, 4096)	16384
bidirectional_3 (Bidirection	(None, 3, 64)	1057024
batch_normalization_14 (Batc	(None, 3, 64)	256
bidirectional_4 (Bidirection	(None, 3, 64)	24832
batch_normalization_15 (Batc	(None, 3, 64)	256
bidirectional_5 (Bidirection	(None, 64)	24832
batch_normalization_16 (Batc	(None, 64)	256
dense_4 (Dense)	(None, 1)	65

Total params: 1,150,685
Trainable params: 1,141,943
Non-trainable params: 8,742

Fig. 13: Architecture of the Conv-LSTM network, the Convolutional part is inside of the time_distributed_1 layer

REFERENCES

[1] S. B. Taieb, G. Bontempi, A. F. Atiya, and A. Sorjamaa, "A review and comparison of strategies for multi-step ahead time series forecasting based on the nn5 forecasting competition," *Expert Systems with Applications*, vol. 39, no. 8, pp. 7067 – 7083, 2012.

[2] J. R. Slawek Smyl and A. Pasqua, "M4 forecasting competition: Introducing a new hybrid es-rnn model." https://eng.uber.com/m4-forecasting-competition/.

[3] Y. Y. H. L. Guokun Lai, Wei-Cheng Chang, "Modeling long- and short-term temporal patterns with deep neural networks," *Carnegie Mellon University*, 2018.

[4] Z. Wang and T. Oates, "Imaging time-series to improve classification and imputation," *CoRR*, vol. abs/1506.00327, 2015.

[5] R. R. Selvaraju, A. Das, R. Vedantam, M. Cogswell, D. Parikh, and D. Batra, "Grad-cam: Why did you say that? visual explanations from deep networks via gradient-based localization," *CoRR*, vol. abs/1610.02391, 2016.