Convolutional Neural Networks, image recognition and financial time series forecasting

Argimiro Arratia[1⋆] and Eduardo Sepúlveda[1]

Computer Science & Faculty of Mathematics and Statistics,
Barcelona Tech (UPC), Spain
argimiro@cs.upc.edu, eduardo.sepulveda@estudiant.upc.edu

Abstract. Convolutional Neural Networks (CNN) are best known as good image classifiers. This model is recently been used for financial forecasting. The purpose of this work is to show that by converting financial information into images and feeding these financial-image representation to the CNN, it results in an improvement in classification.

Keywords: Convolutional neural networks · recurrence plots · time series · forecasting

1 Introduction

Convolutional Neural Networks (CNN), whose first architecture harks back to the model proposed by Yann LeCun et al [5], has been empirically proven to do much better at classifying images (and general high dimension structures) than classical neural networks. A distinctive characteristic of CNN is that it exploits the local spatial coherence of images. By applying convolutions to small squares (or patches) of the input, through a sliding window that runs over the whole image, a CNN learns local features of the image and consequently preserves the spatial relationship between its pixels. Each convolution is determined by a *filter* (or *kernel*) which defines the size of the sliding window to capture the local information (feature extraction) of the image. Applying different filters (and convolutions) the CNN learns different feature maps, thus improving its ability to recognise specific patterns in the input data. For the mathematical details of the CNN model see [4].

In the realm of financial time series forecasting the motivation for using a CNN seems clearly its capability to take advantage of the (local) structural information within the data, ideally by learning feature maps (or their original filters) that represent specific patterns in the time series that is target for future values prediction, such as those extracted from Technical Analysis; or patterns in time-dependent exogenous variables deemed as potential predictors, such as

⋆ Supported by grant TIN2017-89244-R from MINECO (Ministerio de Economía, Industria y Competitividad) and the recognition 2017SGR-856 (MACDA) from AGAUR (Generalitat de Catalunya)

those obtained from Fundamental Analysis (see [1, Ch. 6] for a survey of Technical and Fundamental analysis in financial engineering).

However, the limited picture offered by the 1D nature of time series data could possibly restrict the recognition ability of a CNN. As it has been well studied in signal theory, some features of time series are best identified in the frequency domain rather than in the time domain; likewise, making a higher dimension representation of a time series may give a richer picture of its different features that a CNN can best recognise and take advantage.

In this paper we demonstrate the CNN improvement in financial time series prediction by preprocessing financial data as images before feeding them as inputs to the model. The transformation of numeric data to image is performed through *recurrence plots* (RP).

In the following section we explain the methodology of recurrence plot to transform numeric arrays into a matrix of bits (an image), we further explain details of the convolutional neural network model, and the data sets we use for our financial forecasting experiments. Then in section 3 we report our experiments, which basically consists in comparing the performance in making predictions by a standard CNN versus a CNN endowed with a RP pre-processing unit of the input, and in two different scenarios: one on predicting direction of price of S&P 500 index, and the other on predicting financial distress of a collection of U.S. banks. In section 4 we conclude.

2 Methods and Data

2.1 Recurrence plots

Recurrence plots, introduced by Eckmann et al [3], provides a visualization of the periodic nature of a trajectory through a phase space. Recurrence plots have been used in time series classification [6]. It is formalized as a matrix where each entry is given by the equation

$$R(i,j) = \Theta(\varepsilon - ||\boldsymbol{x}(i) - \boldsymbol{x}(j)||), \quad \boldsymbol{x}(\cdot) \in \mathbf{R}^m, i, j = 1 \dots N \tag{1}$$

where N is the number of states, $\boldsymbol{x}(i)$ is the subsequence observed at time i, $||\cdot||$ is a norm, ε is a threshold for closeness and Θ is the Heaviside function ($\Theta(z) = 0$, if $z < 0$, or 1 otherwise). Thus, if the m-dimensional trajectory of the time series at time i is close (with respect to some metric) to the subsequence observed at time j, there will be a 1 (or a yellow square, as in our image representation) at entry (i, j) of recurrence matrix; otherwise, the value is 0 (a dark purple square).

In our experiments we use the pairwise Euclidean distance for the RP norm $|| \cdot ||$. We observe that one can use RP beyond time series as a general pictorial representation of a similarity metric among multi-dimensional features of some other kind of numeric data apart from time series. We use it in that sense for the bankruptcy classification problem.

2.2 CNN model

Convolutional Neural Networks (CNN) are made up of hidden nodes (or neurons), distributed through various layers, with learnable weights and biases. Each neuron receives several inputs and computes a weighted sum over them, and then passes the result through an activation function which gives an output. Popular choices for activation functions are: the logistic sigmoid ($\sigma(x) = 1/(1+\exp(-x))$), the hyperbolic tangent ($\tanh(x) = 2/(1+\exp(-2x)) - 1$), or the rectified linear units (ReLu, $R(x) = \max(0, x)$). The network's parameters are tuned by minimizing some loss function.

As opposed to regular neural networks, nodes in a CNN are not fully connected but only connected to a local region in the inputs. This local connectivity is attained by using convolutions instead of weighted sums. In each layer of the CNN, inputs are convolved with a weight array (referred as the filter or kernel) to create a feature map; that is, the weight array (kernel) which is of smaller dimension than the input, slides over the input and computes its dot product over each local region. The output of this feature map is then passed through a (non-linear) activation function (e.g. some of the three activation functions mentioned above), and the dimension of this transformed feature map is subsequently reduced by *spatial pooling* (or subsampling). For example, by considering Max, Average or Sum of values. In practice, Max Pooling has been shown to work best. The objective of pooling is to down-sample an input representation, reducing its dimension and allowing for assumptions to be made about features contained in the sub-regions binned.

The procedure is repeated layer by layer (the number of layer determines the *depth* of the network); that is, in each subsequent layer $l = 2, \ldots, D$, the input feature map obtained from previous layer $l-1$, by convolutions and transformation through activation function and pooled, is convolved with a set of M_l kernels to create a new feature map corresponding to the current layer l.

The dimension of the convolution operator accommodates to the dimension of the data. For one-dimensional input, as is the case of time series, $x = \{x(t) : t = 1, \ldots, N\}$, one-dimensional (1D) convolutions with kernels w_h, for $h = 1, \ldots, M$ with $M < N$, can be used, which are formally defined as

$$s(t,h) = (w_h \star x)(t) = \sum_{n=-\infty}^{\infty} x(n) w_h(t-n) \tag{2}$$

For two-dimensional input, as is the case of images, $(I(i,j))_{1 \le i,j \le N}$, the appropriate convolutions must be two-dimensional (2D) on kernel matrices $(K_h(i,j))_{1 \le i,j \le M}$:

$$s(i,j,h) = (K_h \star I)(i,j) = \sum_m \sum_n I(i-m, j-n) K_h(m,n) \tag{3}$$

We use Keras, the Python Deep Learning library[1], for CNN computations. We adopt the nomenclature of Keras, where a 1D CNN is denoted Conv1D, and a 2D CNN is Conv2D.

2.3 Datasets

S&P 500 index. For our stock market prediction experiment the dependent variable or target for prediction is the sign of the Standard & Poors 500 equity premium (GSPCep). This is the difference of logarithms of the price returns, including dividends, and the risk-free interest rate:

$$GSPCep(t) = \log\left(\frac{P(t) + D12(t)}{P(t-1)}\right) - \log(r(t) + 1)$$

where $P(t)$ and $D12(t)$ are, respectively, the price and the 12-month moving sums of dividends paid on the S&P 500 index at time t, and $r(t)$ is the interest rate of the three months U.S. Treasury bill. Our target of prediction is essentially the direction of price with reference to the risk-free interest rate. We consider as exploratory variables or features the past history of the equity premium, up to three lags, and past history of its variance estimated as the square of GSPCep, up to two lags.

This financial data has been retrieved from Amit Goyal's webpage, which builds upon a data collected by Robert Schiller for his statistical analysis of S&P 500. The data is sampled on a monthly basis and ranges from 1990 to 2015.

Bank data. The data consist of arrays with 106 exploratory variables pertaining to financial indicators drawn from quarterly financial reports of U.S. banks, plus 1 response variable which is categorical taking value 1 to indicate a bankrupt entity, or 0 otherwise. There are 5152 of these arrays, each representing the financial situation of a bank. Each bank has at least 8 quarter periods of financial data reporting. This financial information have been retrieved from the Federal Deposit Insurance Corporation (FDIC), who maintains a list of U.S. insured banks that went bankrupt during the period 1992-2017.

3 Experiments and results

The experiments consist in comparing the Convolutional Neural Network's performance with two different processing of its data inputs. The first one consists on feeding the one-dimensional numerical data directly as input to our CNN. The second one consists on first pre-processing the numerical data into images using the Recurrence Plot technique. Each image represents a time series which is categorized according to its response variable. These images provide the input for our CNN.

[1] https://keras.io/

We will compare the performance of the models with respect to accuracy in classification, loss in training, AUC, Matthews Correlation and 10-fold cross-validation. The use of cross-validation in time series data is justified in [2].

We perform the experiments in two different financial scenarios: 1) to predict direction of price of the S&P 500 index; 2) to predict the possibility of bankruptcy in a set of U.S. banks.

3.1 CNN specifications

We build two types of Convolutional Neural Network models using the Keras library in Python. In both cases we used sequential modeling because is the easiest way to build a model layer by layer in Keras. Since our goal is to test the possible enhancement in classification of a CNN with a RP we will not care about tuning in any sophisticated way the inner workings of the CNN and rather work with a basic structure upon which we feed the input processed as image through the recurrence plot method, or not. In the following list we detail the structure of the Convolutional Neural Network in our models and difference between them. The main difference is the pre-processing of data inputs.

Conv1D: Consists of one convolutional layer made of one-dimensional convolutions with kernels of size 2 and 64 nodes. Inputs are numeric arrays. The activation function we use is the ReLU, which works quite well in practice and is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations. The Pooling layer will be produced with Max pooling.

Conv2D + RP: For this model the inputs (arrays of numbers) are first pre-processed with the recurrence plot (RP) method to produce corresponding images (matrices of 0 and 1). Then follows a convolutional layer made of two-dimensional convolutions with kernels of size 2×2 and 64 nodes, a ReLu activation function, and a Max pooling layer.

Compiling the model. Compiling the model takes three parameters: optimizer, loss and metrics.

1. The optimizer controls the learning rate. We will be using "Adam" as our optimizer because it is generally a good optimizer to use for many cases. The Adam optimizer adjusts the learning rate throughout training. The learning rate determines how fast the optimal weights for the model are calculated. A smaller learning rate may lead to more accurate weights (up to a certain point), but the time it takes to compute the weights will be longer.
2. We used *binary crossentropy* for our loss function. This is the most common choice for classification problem with two labels.
3. Metrics: we use the accuracy metric to see the accuracy score on the validation set when we train the model.

Train the model. We will use the "fit()" function with the following parameters: training data (train X), target data (train Y), validation data, epochs number and batch number.

The number of epochs is how many times the entire dataset is passed forward and backward through the neural network. The more epochs we run, the more the model will improve, up to a certain point. After that point, the model will stop improving during each epoch. The batch size is the total number of training examples present in a single batch. For example we can divide the dataset of 60 examples into batches of 12 then it will take 5 iterations to complete 1 epoch. In our experiments we set epoch to 10, and batch to 1.

Validation data. We will use the test set provided to us in our dataset, which we have split into X test and Y test.

3.2 Experiment 1: Predicting direction of SP500

Data preparation. As detailed in section 2.3, the data consists of arrays of 5 exploratory variables (lags 1, 2, and 3 of the GSPCep series and lags 1 and 2 of the square of the series, sampled monthly), plus 1 response variable which is categorical taking value 1 to indicate an increase of the (monthly) price of S&P500, or 0 otherwise. There are 300 of these arrays, representing the multivariate series ranging from 1990 to 2015. We apply a rolling window of size 12 (12 months of data or arrays) to predict next month price direction, and make forward steps of 1 month. Hence, each RP uses 12 periods of financial data.

In the first model (Conv1D) we use as "training X" an input array with the numerical values of the variables of the 12 periods of time and as "training Y" we will use a categorical input array of 0 and 1 for each time series.

On the other hand, for model Conv2D+RP we convert this training (X, Y) numerical input array to image using recurrence plots. In this way we obtain images (a matrix input) which are then fed to the 2D-Convolutional Neural Network.

We can observe in Figure 1 that the images created by recurrence plot present differences when comparing them between their categories (Down = 0 and Up = 1).

Results. We repeat each experiment 100 times and obtain the following average results for each CNN model reported in Table 1.

Table 1. Performance of CNN without/with RP for S&P500 price prediction

	Acc	Loss	10-fold CV	AUC	Matthews cor.
Conv1D	0.52	5.75	61.1% (± 4.16%)	0.65	-0.102
Conv2D + RP	0.63	2.69	63.22% (± 0.93%)	0.66	-0.0026

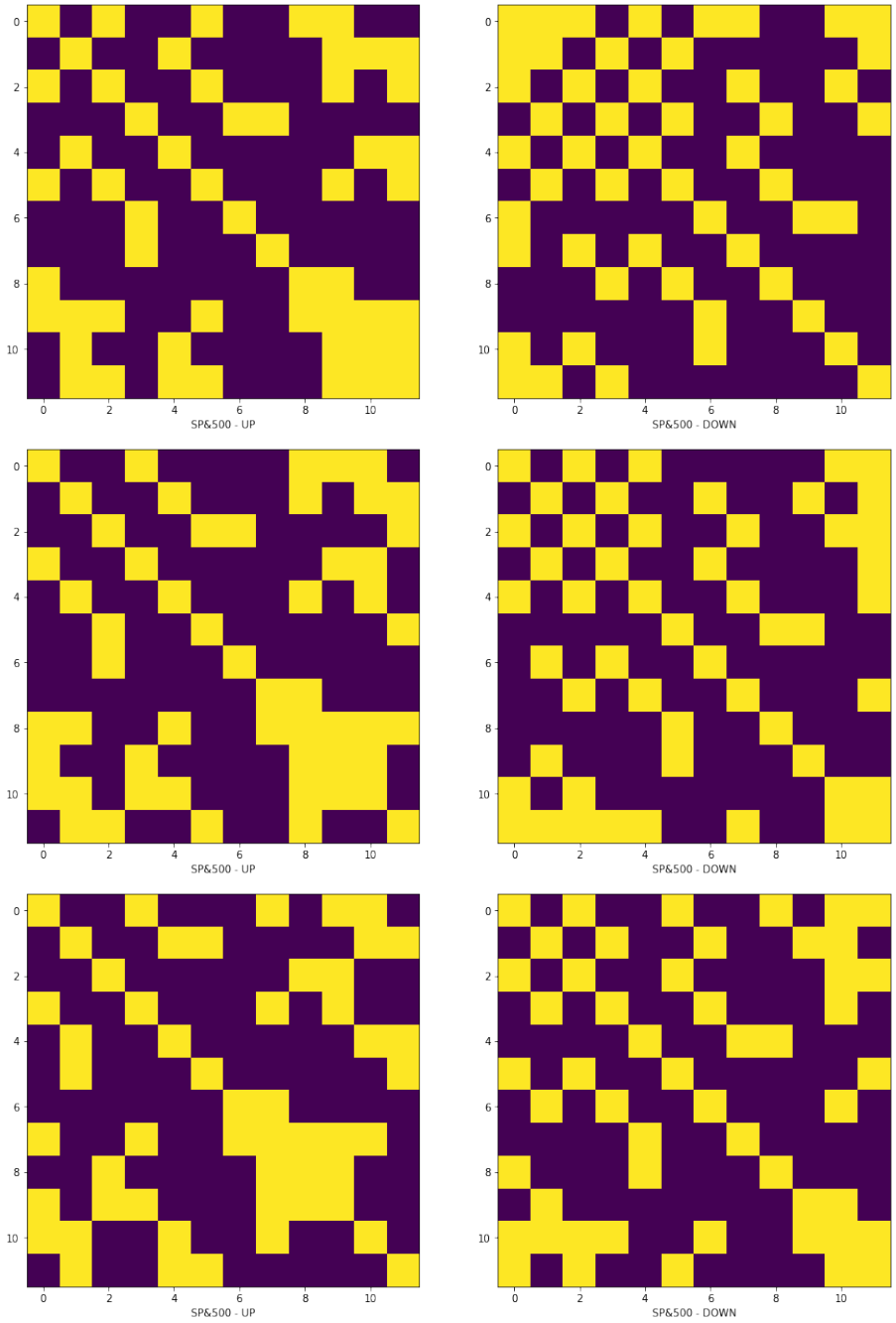

Fig. 1. RP images of S&P 500 data inputs with target 0 (price down) or 1 (price up)

We observed that by making a previous processing of the data by recurrence plot transforming our numerical data to images, we obtain better results consistently improving the accuracy, the AUC, Matthews correlation, while reducing the loss, and the 10-fold CV shows that the combined model Conv2D+RP generalizes well.

3.3 Experiment 2: Bankruptcy Detection

Data preparation. We select the data of eight periods of time (quarters) for each bank and scale the data column by column by min - max method. We added the response variable classifying if the bank will fail or not in the next period of time.

For the first model (Conv1D) we used the input data as stated above for our CNN. But for the second model (Conv2D+RP) we convert the numeric time series data to images (recurrence plot) and give the image as input to our CNN.

In Figure 2 we observe the recurrence plot of three pairs of banks classified as non - bankrupt and another as bankrupt. We can see a clear difference between the graphics which indicates an adequate classification already given by the RP method alone.

Results. Using CNN's with a similar configuration in both cases, and reproducing each experiment 100 times, we obtain the (average) results reported in Table 2.

Table 2. Performance of CNN without/with RP for Bankruptcy Detection

	Acc	Loss	10-fold CV	AUC	Matthews cor.
Conv1D	0.82	3.27	58.91% (± 17.98%)	0.72	0.42
Conv2D + RP	0.94	1.01	93.75% (± 0.07%)	0.83	0.67

The improvement in accuracy, AUC, Matthews correlation and decrease in loss are significant when pre-processing data with RP before feeding it to the 2D Convolutional Neural Network.

4 Conclusions

We have shown that by making a previous processing of the input by recurrence plot transformation of our numerical data to images, we obtain better classification results as measured by five different metrics of classification performance.

We have used a standard norm (Euclidean norm) for the definition of recurrence plots. Other norms are possible, in particular ones more suitable for capturing similarities among financial time series, like, for example, a correlation based metric. It would be interesting to see the difference in performance of

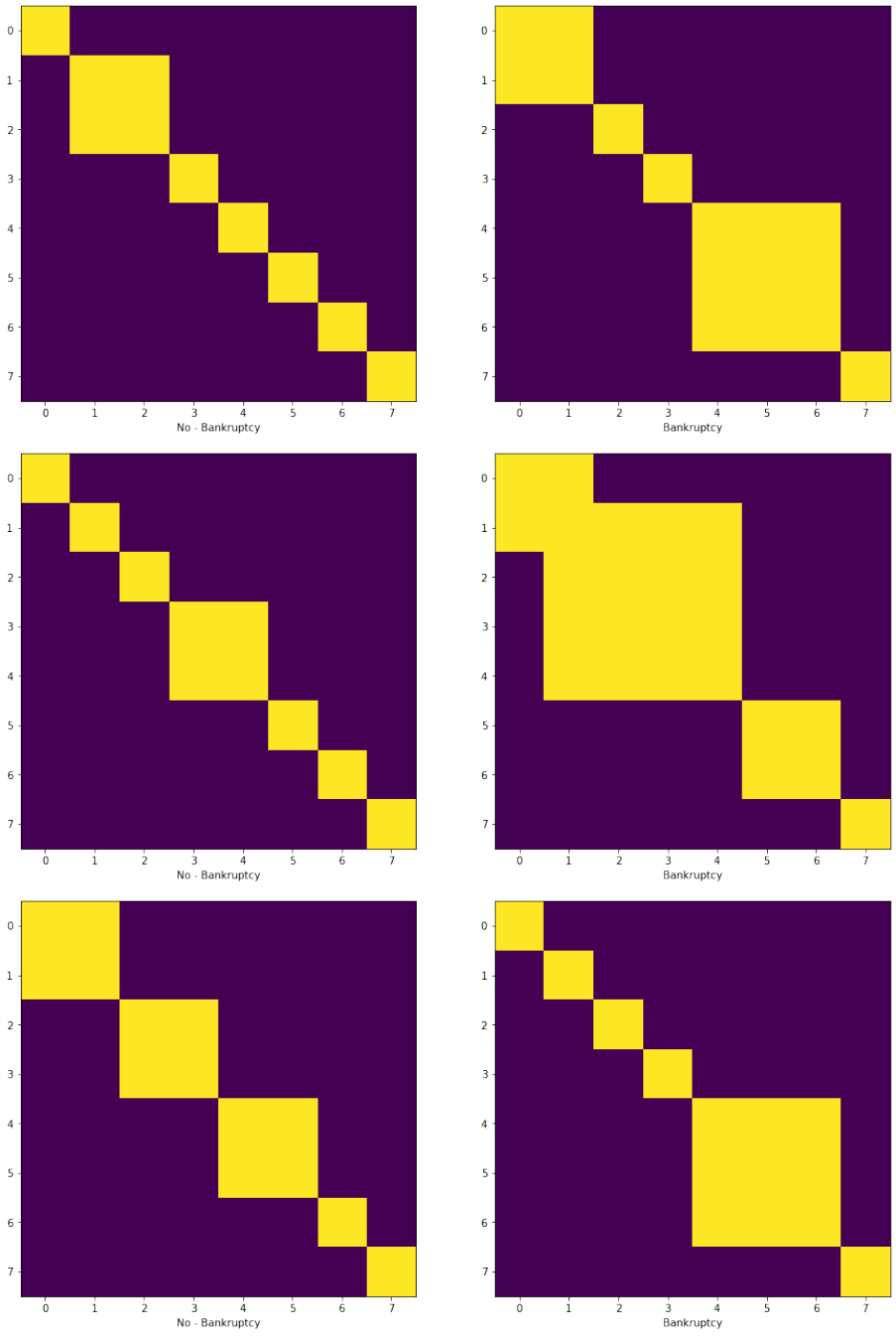

Fig. 2. RP images of U.S. bank data with target 1 (bankrupt) or 0 (non-bankrupt)

Conv2D+RP model under different norms underlying the definition of the RP method.

References

1. Arratia, A. (2014). *Computational Finance: An Introductory Course with R*. Atlantis Press
2. Bergmeir, C., Hyndman, R. J., Koo, B. (2018). A note on the validity of cross-validation for evaluating autoregressive time series prediction. *Computational Statistics and Data Analysis*, 120, 70-83.
3. Eckmann,J. P., Kamphorst, S. O., Ruelle, D. (1987). Recurrence Plots of Dynamical Systems. *Europhysics Letters*. 5 (9): 973-977.
4. Goodfellow, I., Bengio, Y., Courville, A. (2016). *Deep Learning*. The MIT Press.
5. LeCun, Y., Bottou, L., Bengio, Y., Haffber, P. (1998). Gradient-based learning applied to document recognition. *Proc. of the IEEE*, 86, 2278-2324.
6. Silva, D. F., De Souza, V. M., Batista, G. E. (2013). Time series classification using compression distance of recurrence plots. *In: 2013 IEEE 13th International Conference on Data Mining* 687-696, IEEE.