Exploring the RCNN technique in a multiple-point statistics framework[1]

Sebastian Avalos (sebastian.avalos@queensu.ca)
Julian M. Ortiz (julian.ortiz@queensu.ca)

Abstract

This work explores the Recursive Convolutional Neural Network (RCNN) technique in terms of describing (1) the inner architecture, (2) the training process and (3) the simulation algorithm in a multiple-point statistics simulation framework. To acquire a more intuitive understanding of the previous description, visualizations of hidden layers activations using three different training images is presented. Two interesting applications (1) using RCNN E-types as secondary information for MPS algorithms and (2) training RCNN with scarce and limited information, are also explored.

1. Introduction

Deep Learning techniques (Goodfellow et al., 2016), a set of Deep Neural Networks (DNN), have enriched the geoscientist toolkit during the last few years. Finding applications of DNN in geostatistics has gained the researchers' attention. While some authors propose the integration of DNN as secondary elements inside accepted workflows, others are exploring the possibilities of new workflows built only up by DNN elements. This is reflected particularly in the multiple-point statistics (MPS) simulation framework where partially integrated DNN workflows are presented by Azamifard et al. (2019) using Convolutional Neural Networks (CNN) and by Hashemi et al. (2014) using Feedforward Neural Networks, while fully DNN workflows are presented by Mosser et al. (2017); Laloy et al. (2018) use Generative Adversarial Neural networks, Laloy et al. (2017) use CNN coupled with Variational Autoencoders and Avalos and Ortiz (2019a,b) use nested CNNs.

In this work, we explore the Recursive Convolutional Neural Network (RCNN) technique (Avalos and Ortiz, 2019a,b). The RCNN corresponds to a set of N nested CNNs, all trained with the same Training Image, in such a way that the CNN$_i$ predicts a pattern based on a very small number of conditional nodes and the previously predicted patterns from CNN$_{i-1}$,..., CNN$_1$. How to convert the training image in a training dataset, the RCNN architecture and the training and simulation processes are described in section 2. Understanding the learned features by means of hidden layer visualizations, boosting other MPS methods with the RCNN E-type as probability map (soft data), and ideas/discussions of using RCNN without training image are the purpose of section 3. Lastly, a brief summary ends the present work.

2. Describing the RCNN technique

In this section the RCNN is described in terms of (1) the building process of the training dataset from a training image, (2) the building blocks of the RCNN architecture, and (3) the training and simulations processes.

[1]Cite as: Avalos S, Ortiz JM (2019) Exploring the RCNN technique in a multiple-point statistics framework, Predictive Geometallurgy and Geostatistics Lab, Queen's University, Annual Report 2019, paper 2019-03, 30-40.

2.1. Training image as training dataset

Training images (TI), a key assumption on any multiple-point statistic methods, are assumed to contain a variety of possible patterns distributed in accordance with the phenomenon to be modelled. For instance, Figure 1 shows three different TIs used throughout this work.

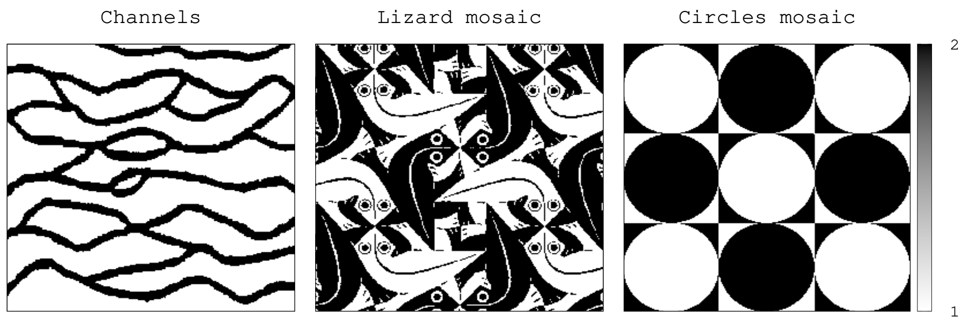

Figure 1: Training images used in this work. (a) Channels, (b) lizard mosaic and (c) circle mosaic.

Let SG and IP be the collocated search grid and inner pattern. Users decide their dimensions, being suggested the use of odd values. From the TI, all pairs SG $\leftrightarrow$ IP are extracted. Over each pair, a small amount of known locations at the SG are extracted ($0.1\% - 5.0\%$), defining the so-called data event (*dev*). The main aim is to train the RCNN using *dev* as input to predict the IP. To avoid unnecessary data storage, the $\{dev, IP\}$ tuples are created, used and erased at each epoch (cycle) during the training process and only the source SG $\leftrightarrow$ IP are kept to provide the next $\{dev, IP\}$ tuples for the next epoch.

2.2. The RCNN architecture

We begin by describing a conventional CNN (Figure 2. Left), which is composed of a feature extraction section and a classification section. Let $\mathbf{X}$, $\mathbf{W}_m$, $\mathbf{b}_m$ and $\mathbf{H}_m$ be the input, filters, bias vectors and hidden layers, respectively in the feature extraction section. $\mathbf{H}_m$ results of convolving $\mathbf{W}_m$ over $\mathbf{H}_{m-1}$, adding a bias vector $\mathbf{b}_m$ and passing the result through (1) a Batch-Normalization function, (2) a non-linear activation function and (3) a pooling function (Equation 1).

$$\mathbf{H}_m = \begin{cases} \mathbf{X} & \text{if } m = 0 \\ pool(g(BN(\mathbf{W}_m\mathbf{H}_{m-1} + \mathbf{b}_m))) & \text{if } 0 < m \leq M \end{cases} \tag{1}$$

Convolution The process of sliding filters over the input while performing the sum of an element-wise multiplication between the filter values and the values of the corresponding input section.

Activation function $g(\cdot)$. A non-linear transformation applied element-wise over each hidden layer. It has an active zone where its derivative is not zero. A rectified linear unit (ReLU) function is used in this work.

Batch Normalization $BN(\cdot)$. It normalizes the convolution results before passing them through the activation function. It subtracts the mean and divides by the respective standard deviation, over each feature map (the convolving result of each filter).

Pooling $pool(\cdot)$ It performs a downsampling of each feature map. Here, the *max pooling* function is used, which maintains the maximum value of zone of 2×2 with a stride of 2×2.

In the classification section, $\mathbf{W}_{FC_f}$, $\mathbf{b}_f$ and $\mathbf{FC}_f$ are the weight matrices, bias vector and hidden fully connected layer, respectively. $\mathbf{FC}_f$ results of a matrix multiplication between $\mathbf{W}_{FC_f}$ and $\mathbf{FC}_{f-1}$, adding a bias vector $\mathbf{b}_f$ and passing the result through (1) a Batch-Normalization function and (2) an activation function (Equation 2).

$$\mathbf{FC}_f = \begin{cases} Vec(\mathbf{H}_M) & \text{if } f = 0 \\ g(BN(\mathbf{W}_{FC_f}\mathbf{FC}_{f-1} + \mathbf{b}_f)) & \text{if } 0 < f \leq F \end{cases} \tag{2}$$

The first feedforward network $\mathbf{FC}_0$ corresponds to a vector-representation $Vec(\mathbf{H}_M)$ of the last hidden layer $\mathbf{H}_M$. The last layer $\mathbf{FC}_F$ passes through a softmax function to obtain the expected probability for each category on a categorical prediction. The predicted category is selected based on the maximum probability value.

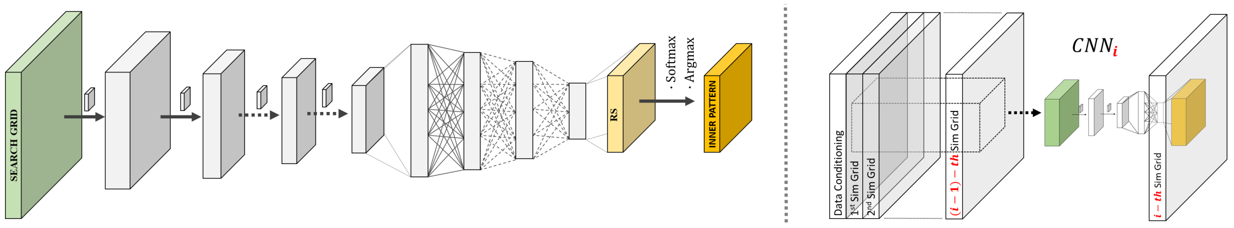

Figure 2: Architectures. (Left) Conventional CNN. (Right) Scheme of the i^{th} nested CNN in a RCNN *architecture.*

The idea behind the RCNN technique is to nest several CNNs so the CNN$_i$ is trained to predict the closest pattern for a given data event concatenated with the previously simulated patterns from CNN$_{i-1}$,...,CNN$_1$ (Figure 2. Right). Note that the simulated pattern differs from the predicted pattern since, once predicted, only a certain percentage of the pattern is actually frozen on the simulation grid (see subsection 2.4). The purpose behind the recursiveness is the improvement of results quality by taking into account the previously simulated information.

2.3. Training

Let D^i, with i $0, .., N$, be the i^{th} domain (simulation grid), with the size of the TI. Randomly, a small amount of conditioning data is extracted from the TI and assigned to each domain. The simulation process (see subsection 2.4) is performed on each domain, except for D^0. The CNN$_i$ simulates D^i. The data base (DB^i), inputs $\leftrightarrow$ outputs ($\mathbf{X}^i \leftrightarrow$ IP^{Real}), is extracted from the simulated domains to train each CNN$_i$. Here, $\mathbf{X}^i$ represents the concatenated input $[SG(D^0), SG(D^1), ..., SG(D^{i-1})]$ collocated with IP^{Real} (known from the TI). The loss function (Equation 3) to be minimized during training corresponds to the sum of all cross entropies between the predicted IP and the real IP. It makes use of the predicted probability $\hat{p}(k)$ and the real probability $p(k)$, coming from IP^{Real}, represented as a vector of [0.. 1 ..0] with 1 at position k. This reduces the complexity of Equation 3 when calculated.

$$\mathcal{L}_i(\Theta^i) = -\sum_{(a,b)\ \ IP} \sum_{k=1}^{K} \Big(\log \hat{p}\big(k|\mathbf{RS}_{a,b}; (\mathbf{X}^i, \Theta^i)\big)\Big) \cdot p\big(k|\text{IP}^{Real}_{a,b}; (\mathbf{X}^i)\big) \tag{3}$$

Inner parameters $\Theta = \{\mathbf{W}, \mathbf{b}\}$ are initialized as random values from a normal gaussian truncated function. Each CNN$_i$ receives a mini-batch of m samples from DB^i, estimates the gradient with respect to the loss function, ${}_{\Theta}\mathcal{L}_i(\Theta^i) \approx {}_{\Theta}\big[1/m \sum_{k=1}^{m} \mathcal{L}_i\big(\Theta^i; \mathbf{X}^i_k\big)\big]$, and performs a parameter update $\Theta^i_t \leftarrow \Theta^i_t +\ \ \Theta^i$ by inferring the updated direction $\ \Theta^i$ with respect to the gradient ${}_{\Theta}\mathcal{L}_i\big(\Theta^i\big)$ by using the Adam Optimizer (Kingma and Ba, 2014). After all CNN$_i$ have been trained by all mini-batches, the first epoch is completed. In this work, all RCNN are trained with 400 epochs with a fixed learning rate of $3 \cdot 10^{-3}$.

2.4. Simulation

The available conditioning data migrate to the closest nodes at each D^i. Every domain is simulated, starting on D^1 and ending with D^N. A single random path is previously defined and then used for all domains. Along the path, and at every unknown location, the $\mathbf{X}^i$ matrix is fed into the CNN$_i$ to obtain the predicted IP^i. Then, instead of freezing all IP^i values in D^i, only a random percentage of them are selected and frozen at unknown locations. Particularly, the unknown center is always simulated. The percentage of random values used in this work is 50%.

3. Exploring the RCNN technique

In this section the RCNN is analyzed once trained by means of (1) visualizing the activity at the hidden layers, (2) a simple workflow to boost other MPS technique with RCNN, and (3) training the RCNN with only scarce information without training image.

3.1. Visualization of hidden layers

The usual approach to visualize feature maps in the hidden layers is to project back their values to the input space (Zeiler and Fergus, 2014). This is valid and appropriate when the CNN is trained with fully informed images as ImageNet (Deng et al., 2009), where hierarchical features are implicit on each image of the set. However, this approach seems not suitable here since each input corresponds to a small amount of conditioning data embedded in a matrix of zeros (unknown nodes) and previously simulated patterns. To deal with this issue, we propose to begin by a direct visualization of activity of inner neurons (hidden layers) when fed by different input data.

We trained three different RCNN using the conditions depicted in subsection 2.3, each one with one of the training images of Figure 1. Each RCNN has a SG/IP of $19 \times 19/19 \times 19$ and four nested CNNs and four hidden layers, each one with 16 feature maps. Figure 3, Figure 4 and Figure 5 illustrate the activities of each feature map at each hidden layer for two different inputs (data events and the respective simulated patterns) at the last CNN in the trained RCNN.

The first hidden layer (HL 1) has feature maps that react completely to the presence of structures, others react to the background and a few others almost replicate the data event plus some small noise. These seem to decompose the input data into uncorrelated structures, mimicking other statistical processes that transform raw data into uncorrelated factors. As we move onto deeper layers their sizes are reduced due to the max pooling function. The HL 2 intensifies the input decomposition. HL 3 seems to start an encoding process ending with a HL 4 behaving as a digital fingerprint of the raw input.

Note that the output (predicted pattern) is different to the target pattern when the latter has pointed and linear shapes, and is similar for rounded shapes. Also, small details (1 to 3 nodes in extension) are hardly captured and reproduced, possibly attributed to the use of square filters and pooling functions.

We can use visualizations to validate the use of certain functions in the inner architecture. For instance, Figure 6 illustrates the activity on hidden layers when Batch Normalization is not included. In HL 1, eight over sixteen feature maps do not react to the input stimulus, five of them barely react but in a fuzzy manner and only three respond but in a similar fashion. This can be interpreted as (1) there is no input decomposition due to the lack of a regularization technique (as Batch Normalization does), and (2) most neurons in the first hidden layer are so-called *dead neurons* due to vanishing gradient problem during training, decreasing considerably the effectiveness of the entire network in terms of minimizing Equation 3. The subsequent hidden layers show similar behaviours as HL 1. In particular, HL 4 now does not seem to act as a digital fingerprint of a learnt pattern, reflected in a predicted pattern (output) that is not accurate with respect to the target pattern as shown when Batch Normalization is included.

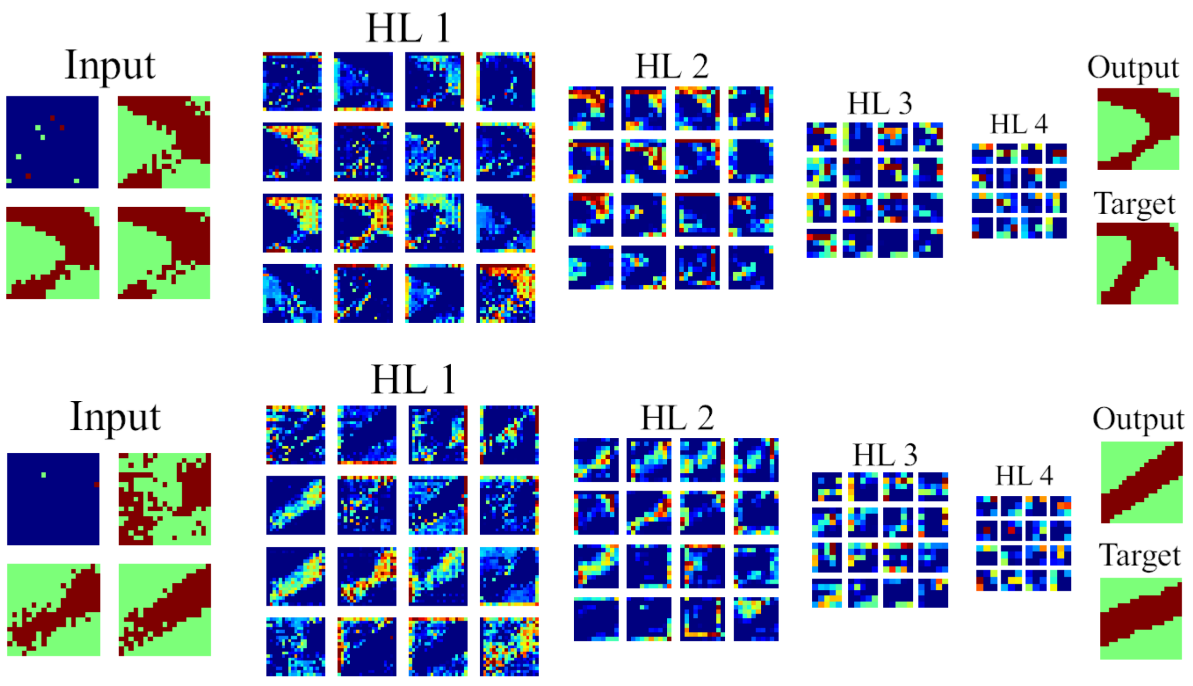

Figure 3: Channels. Activation zones on features maps at each hidden layer feeding two different data events as input.

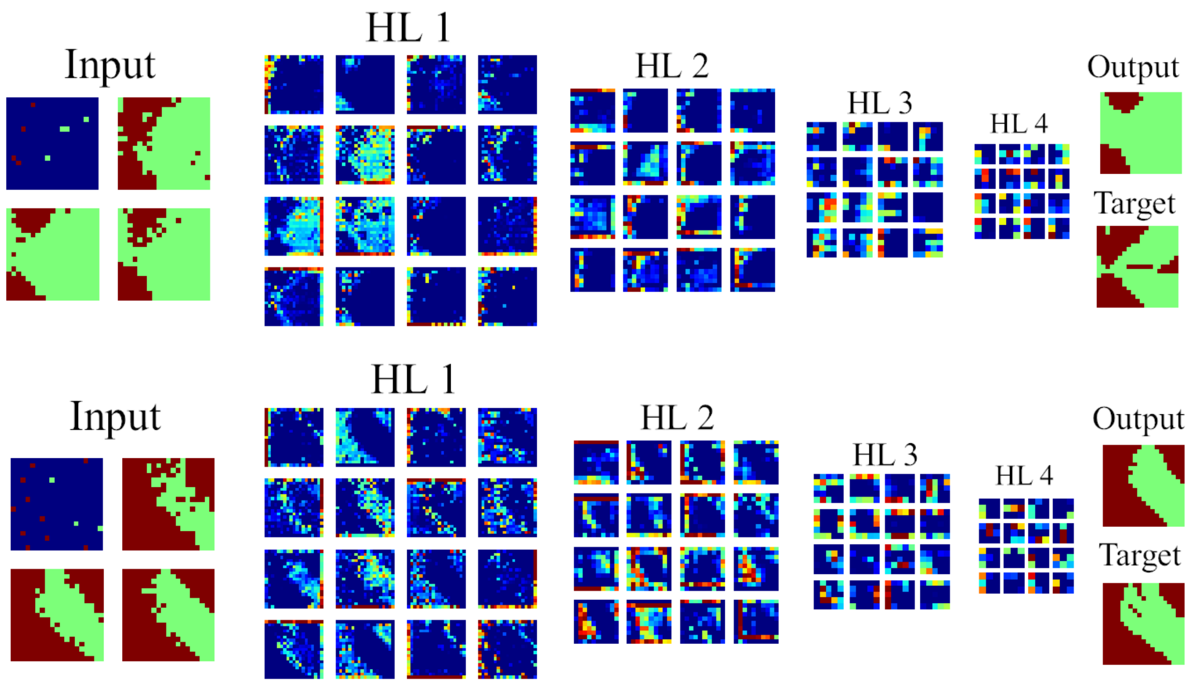

Figure 4: Lizards mosaic. Activation zones on features maps at each hidden layer feeding two different data events as input.

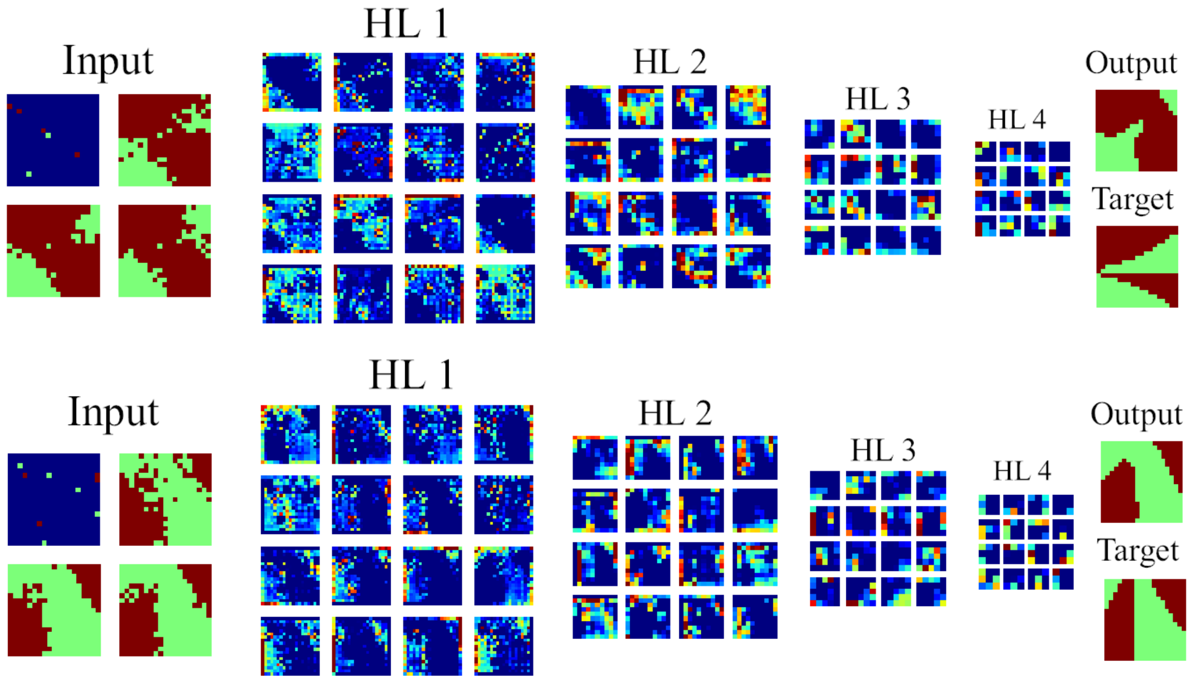

Figure 5: Circles mosaic. Activation zones on features maps at each hidden layer feeding two different data events as input.

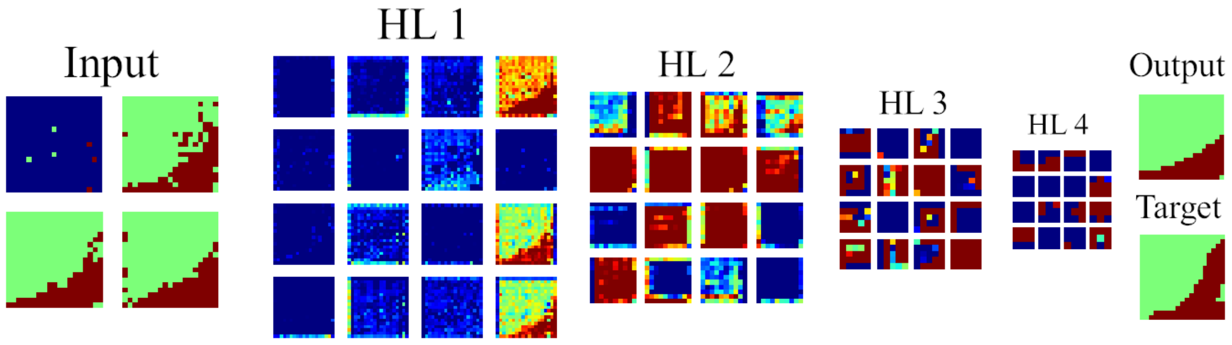

Figure 6: Channels. Activation zones on features maps at each hidden layer when Batch Normalization is not included in the RCNN *architecture.*

Now, if neither the Batch Normalization nor the Max Pooling (2×2) functions are included (Figure 7) the previous analysis is amplified in terms of a lack of pattern recognition during the learning process of *inputs* ↔ *outputs* ($\mathbf{X}^i \leftrightarrow \mathrm{IP}^{Real}$). Indeed, only four of the sixteen features maps in HL 1 seem to be activated to the conditioning data presence, five to the previously simulated patterns and the rest seems to have dead neurons. The last hidden layer (HL 4) is composed of either fully activated or inert feature maps. Lastly, only the direction of the target pattern is predicted but shapes are poorly reproduced.

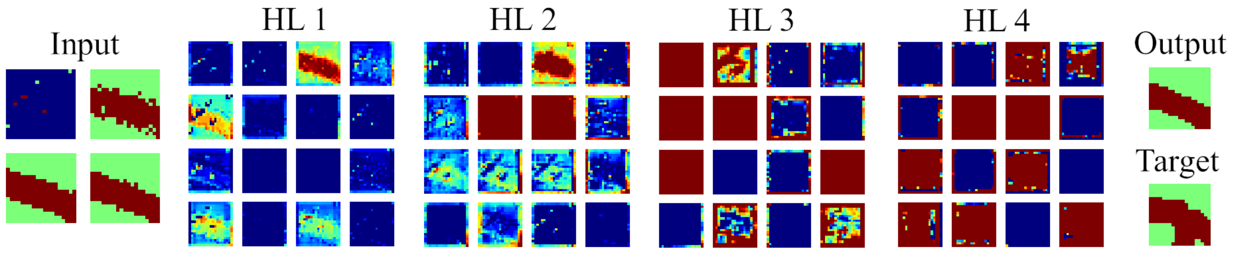

Figure 7: Channels. Activation zones on features maps at each hidden layer when neither Batch Normalization nor Max Pooling are included in the RCNN *architecture.*

3.2. Using RCNN E-type as secondary information in MPS

The RCNN is trained to provide the most probable class/category at each node during simulations. As shown previously (Avalos and Ortiz, 2019a,b) this leads to a lack of reproduction in the probability of classes, at each realization, but to a reasonable probability map when the E-type is considered over a certain number of possible scenarios. Therefore, it is interesting to merge RCNN with any other classic MPS technique (Figure 8) so the former provides an exhaustive probability map while the latter integrates this conditional probability as *soft data*, improving the quality of its realizations.

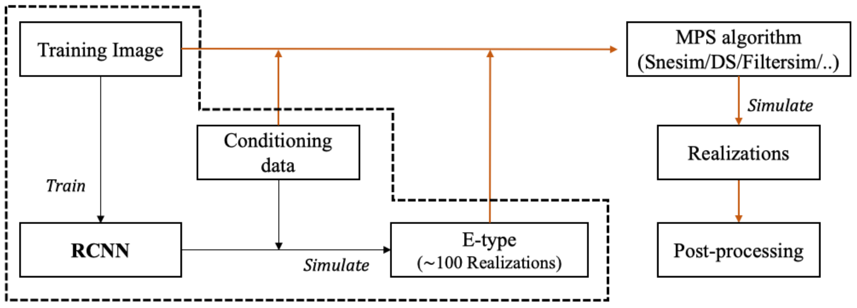

Figure 8: Diagram. Integration of RCNN *E-type as soft data for any other MPS algorithm.*

In probability aggregation, the posterior probability when the conditioning data and the local probabilities from the RCNN are considered independent of each other, is directly the product of their probabilities. This is mostly unusual so redundancy must be taken into account. There are some approaches to tackle this problem. Bordley (1982) proposes a general formula using the respective odd ratios for each probability known as the Brodley's formula. The tau-nu model (Journel, 2002; Polyakova and Journel, 2007), based also on their odd ratios, has a straightforward expression when dealing with binary variables that is also a particular case of the Broadley's formula.

Let $\mathbf{u}$ be an unknown location and $P_a[Z(\mathbf{u}) = c]$ the final aggregated probability for category c . Let $P_{TI}[Z(\mathbf{u}) = c]$, $P_{MPS}[Z(\mathbf{u}) = c]$ and $P_{RCNN}[Z(\mathbf{u}) = c]$ be the TI a-priori probability, the MPS probability without accounting for secondary information and the probability coming from the E-type RCNN respectively, then the expressions are related as follows:

$$
\begin{aligned}
P_a[Z(\mathbf{u}) = c] &= \frac{1}{1+x} \\
\frac{x}{x_0} &= \left(\frac{x_1}{x_0}\right)^{\tau_1} \left(\frac{x_2}{x_0}\right)^{\tau_2} \\
x_0 = \frac{1 - P_{TI}[Z(\mathbf{u}) = c]}{P_{TI}[Z(\mathbf{u}) = c]} \quad , \quad x_1 &= \frac{1 - P_{MPS}[Z(\mathbf{u}) = c]}{P_{MPS}[Z(\mathbf{u}) = c]} \quad , \quad x_2 = \frac{1 - P_{RCNN}[Z(\mathbf{u}) = c]}{P_{RCNN}[Z(\mathbf{u}) = c]}
\end{aligned}
\tag{4}
$$

where Mariethoz and Caers (2014) suggest $\{\tau_1, \tau_2\} = \{1, 1\}$ assuming a standardized form of conditional independence. A similar approach was used by Hashemi et al. (2014) but using $\{\tau_1, \tau_2\} = \{1, 3\}$ for an MPS probability prediction and neural network prediction, respectively. For notation simplicity we replace $P[Z(\mathbf{u}) = c]$ by P and propose a slight modification of Equation 4 by adding $\epsilon = 10^{-3}$ to avoid divisions by zeros when $P = 0$ or resulting on zeros when $P = 1$, as:

$$
x_0 = \frac{1 - P_{TI} + \epsilon}{P_{TI} + \epsilon} \quad , \quad x_1 = \frac{1 - P_{MPS} + \epsilon}{P_{MPS} + \epsilon} \quad , \quad x_2 = \frac{1 - P_{RCNN} + \epsilon}{P_{RCNN} + \epsilon} \tag{5}
$$

and we propose to redefined $\{\tau_1, \tau_2\}$ according to the problem context.

Let's have, for instance, an image reconstruction problem where the training image represents a lizards mosaic. The image is reconstructed from 1%, 2% and 5% of known information. The Direct Sampling (DS) method (Mariethoz et al., 2010) is selected to illustrate the previous idea.

In here, the a-priori probability of a black pixel is $P_{TI} = 0.54$. As DS method searches for the closest pattern on the TI delivering only the found category, the associated probability for that category will always be 1. For instance, if a black pixel is found then $P_{MPS} = 1$ while the RCNN E-type value ($PrMap$) accounts for $P_{RCNN} = PrMap$. In the case where DS found a white pixel value, the P_{MPS} is again assumed to be 1, but the $P_{TI} = 1 - 0.54 = 0.46$ and $P_{RCNN} = 1 - PrMap$.

To obtain the final aggregate probability P_a, the $\{\tau_1, \tau_2\}$ values must be defined. To finally decide if the MPS prediction is accepted or rejected, either (1) a uniform random number is drawn and compare with P_a or (2) a threshold value (t_v) over P_a can be previously defined. The latest is used here. The corresponding values are $\{\tau_1, \tau_2\} = \{0.5, 1.5\}$ and $t_v = 0.95$ (on both predicted categories by DS). All these internal parameters $\{\tau_1, \tau_2, t_v\}$ should be tuned according to the context problem and the expected influence by the RCNN E-type map.

Results are shown in Figure 10. The DS inner parameters are omitted for simplicity but the number of samples considered during simulation is 8. The incorporation of the RCNN E-type maps leads to a clearer delimitation for both categories with randomness only on their boundaries.

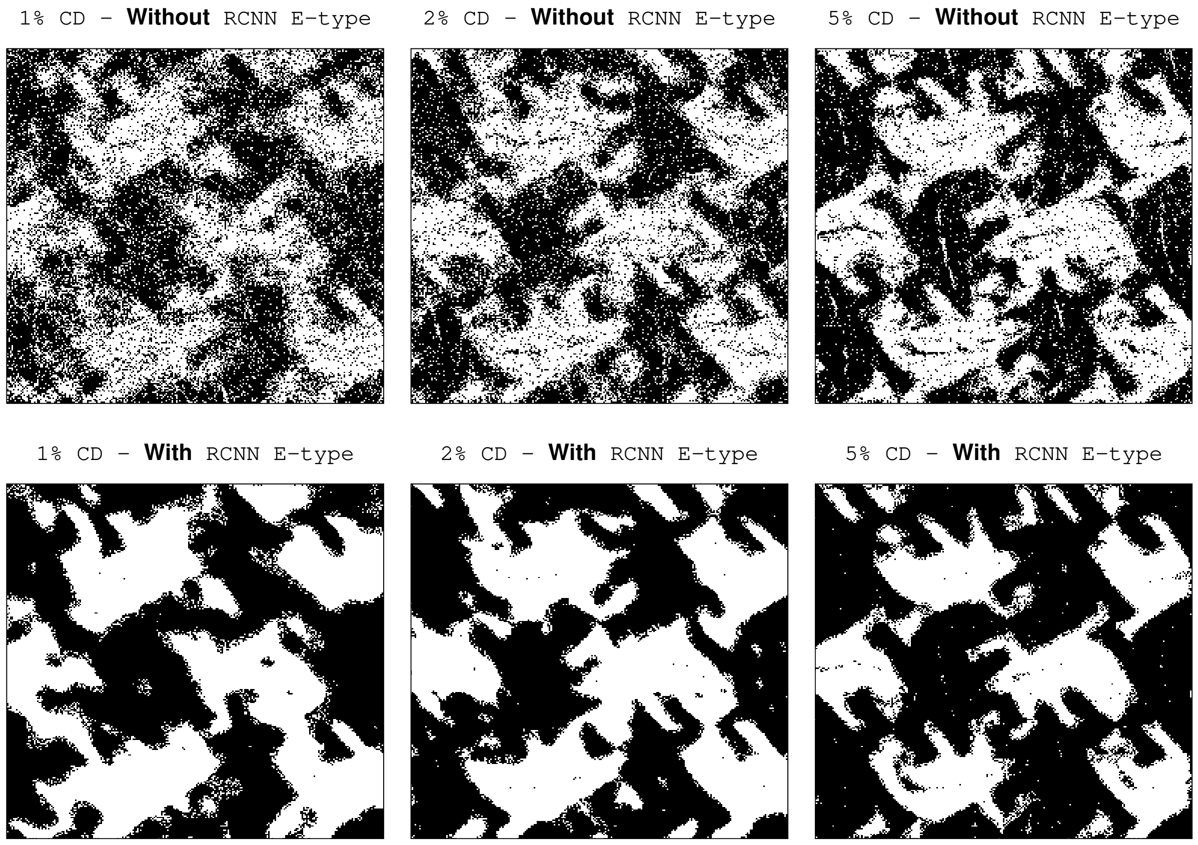
1% CD - Without RCNN E-type
2% CD - Without RCNN E-type
5% CD - Without RCNN E-type
1% CD - With RCNN E-type
2% CD - With RCNN E-type
5% CD - With RCNN E-type

Figure 9: Realizations without (top) and with (bottom) the integration of the RCNN *E-type map.*

3.3. Conditioning data as the only source of information

There is an interesting potential for Neural Networks, particularly for RCNN, in presence of rich conditional information (in terms of quantity of samples and quality of their information). The fact of extracting the training database from one or several training images, as shown in subsection 2.1, is due to the widely known necessity of DNN architectures of having a large set of training information. It compensates the lack of generalization observed on DNN when they are trained over a small amount of information. Therefore, training any DNN using only conditioning data (CD) presents other challenges. Here, we show a simple application of RCNN trained only on sample information.

We use the same RCNN architecture and training conditions as before except for the IP size that now is 1×1, which means this is a pixel-based simulation, rather than patch-based. As the amount of training data is highly reduced in this approach, an early stopping criterion during training is required to avoid overfitting. Indeed, the training process is stopped with 35 epoch instead of 400. This is an arbitrary decision based on experimental results. Results of training three different RCNN with 1%, 2% and 5% of conditioning data are shown in Figure 10. As the ground truth is known (Figure 1), it is possible to recognize that shapes and connectivities are reasonably reproduced with 5% of CD but they get fuzzy as the number of samples decreases. Interestingly, channels are only predicted close to channel samples and not randomly across the domain, meaning that at least a certain understanding of the channel structure is captured during training.

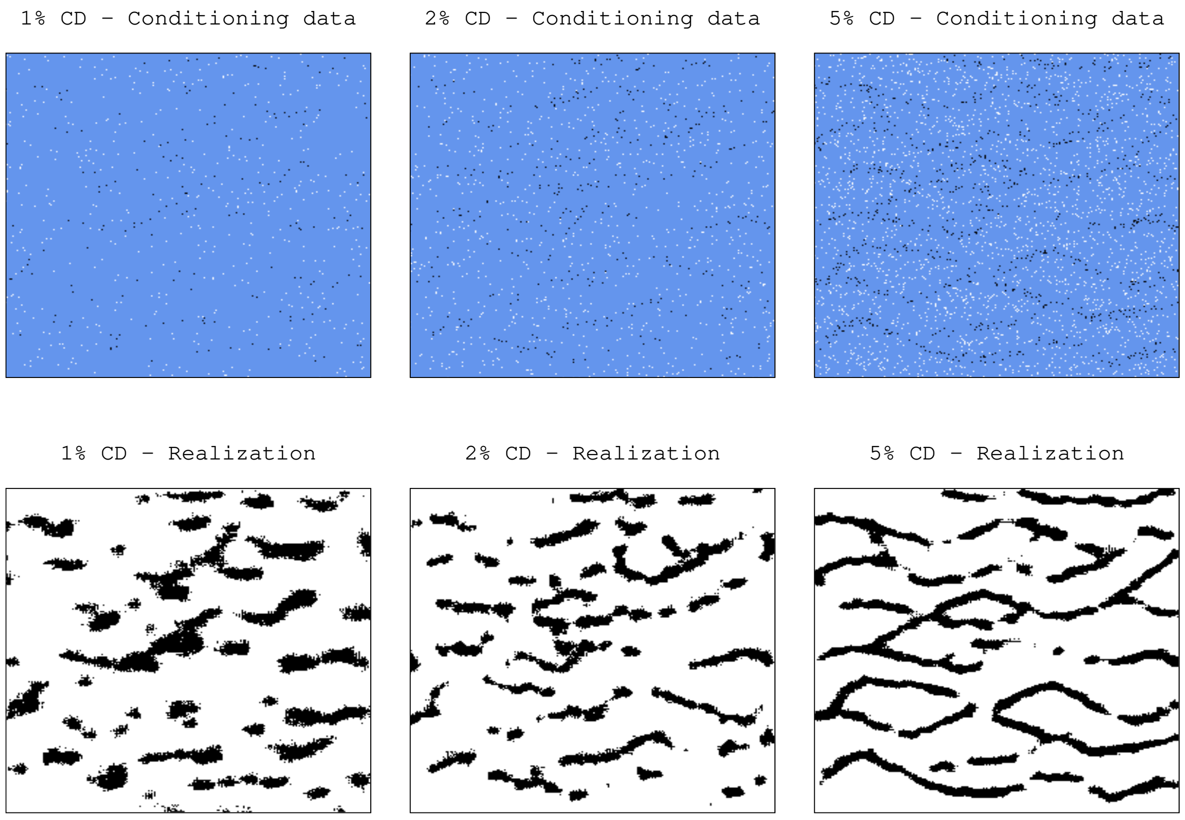

Figure 10: Channels. Conditioning data (top) and realizations (bottom) using 1%, 2% and 5% of CD.

4. Summary

The Recursive Convolutional Neural Network (RCNN) was briefly described in terms of architecture, training and simulation algorithm. By visualizing the activation of feature maps in the RCNN hidden layers, the need of Batch Normalization and Max Pooling functions in the RCNN architecture was proven. The first one serves as a regularization technique while the second one serves as another non-linear function to create a more accurate input representation for further predictions.

Boosting other MPS techniques, as Direct Sampling, by using the RCNN E-types as secondary information (probability maps) was shown. To merge their probabilities the tau-nu model is proposed, tuning the E-type influence with the model parameters. Finally, the RCNN was trained only with scarce sample information to illustrate the difficulties and potential of neural network in a pattern recognition (classification) task using a low amount of information.

5. Bibliography

Avalos, S., Ortiz, J. M., 2019a. Geological modeling using a recursive convolutional neural networks approach. arXiv preprint arXiv:1904.12190.

Avalos, S., Ortiz, J. M., 2019b. Recursive convolutional neural networks in a multiple-point statistics framework. In: Mining goes Digital: Proceedings of the 39th International Symposium “Application of Computers and Operations Research in the Mineral Industry” (APCOM 2019), June 4-6, 2019, Wroclaw, Poland. CRC Press, p. 168.

Azamifard, A., Rashidi, F., Ahmadi, M., Pourfard, M., Dabir, B., 2019. Toward more realistic models of reservoir by cutting-edge characterization of permeability with MPS methods and deep-learning-based selection. Journal of Petroleum Science and Engineering 181, paper 106135.

Bordley, R. F., 1982. A multiplicative formula for aggregating probability assessments. Management science 28 (10), 1137–1148.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., Fei-Fei, L., 2009. Imagenet: A large-scale hierarchical image database. In: 2009 IEEE conference on computer vision and pattern recognition. IEEE, pp. 248–255.

Goodfellow, I., Bengio, Y., Courville, A., 2016. Deep learning. Vol. 1. MIT press Cambridge.

Hashemi, S., Javaherian, A., Ataee-pour, M., Tahmasebi, P., Khoshdel, H., 2014. Channel characterization using multiple-point geostatistics, neural network, and modern analogy: A case study from a carbonate reservoir, southwest Iran. Journal of Applied Geophysics 111, 47–58.

Journel, A., 2002. Combining knowledge from diverse sources: An alternative to traditional data independence hypotheses. Mathematical Geology 34 (5), 573–596.

Kingma, D. P., Ba, J., 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Laloy, E., Hérault, R., Jacques, D., Linde, N., 2018. Training-Image Based Geostatistical Inversion Using a Spatial Generative Adversarial Neural Network. Water Resources Research 54 (1), 381–406.

Laloy, E., Hérault, R., Lee, J., Jacques, D., Linde, N., 2017. Inversion using a new low-dimensional representation of complex binary geological media based on a deep neural network. Advances in water resources 110, 387–405.

Mariethoz, G., Caers, J., 2014. Multiple-point geostatistics: stochastic modeling with training images. John Wiley & Sons.

Mariethoz, G., Renard, P., Straubhaar, J., 2010. The direct sampling method to perform multiple point geostatistical simulations. Water Resources Research 46 (11), w11536.

Mosser, L., Dubrule, O., Blunt, M. J., 2017. Reconstruction of three-dimensional porous media using generative adversarial neural networks. Physical Review E 96 (4), 043309.

Polyakova, E. I., Journel, A. G., 2007. The nu expression for probabilistic data integration. Mathematical Geology 39 (8), 715–733.

Zeiler, M. D., Fergus, R., 2014. Visualizing and understanding convolutional networks. In: European conference on computer vision. Springer, pp. 818–833.