Speak so a physicist can understand you! TetrisCNN for detecting phase transitions and order parameters

Kacper Cybiński
Faculty of Physics, University of Warsaw, Pasteura 5, 02-093 Warsaw, Poland

James Enouen
Department of Computer Science, University of Southern California,
Los Angeles, CA 90089, USA

Antoine Georges
Center for Computational Quantum Physics, Flatiron Institute,
162 Fifth Avenue, New York, NY 10010, USA
Collège de France, 11 place Marcelin Berthelot, 75005 Paris, France
CPHT, CNRS, École Polytechnique, IP Paris, F-91128 Palaiseau, France
DQMP, Université de Genève, 24 quai Ernest Ansermet, CH-1211 Genève, Switzerland

Anna Dawid[*]
Center for Computational Quantum Physics, Flatiron Institute,
162 Fifth Avenue, New York, NY 10010, USA
adawid@flatironinstitute.org

Abstract

Recently, neural networks (NNs) have become a powerful tool for detecting quantum phases of matter. Unfortunately, NNs are black boxes and only identify phases without elucidating their properties. Novel physics benefits most from insights about phases, traditionally extracted in spin systems using spin correlators. Here, we combine two approaches and design TetrisCNN, a convolutional NN with parallel branches using different kernels that detects the phases of spin systems and expresses their essential descriptors, called order parameters, in a symbolic form based on spin correlators. We demonstrate this on the example of snapshots of the one-dimensional transverse-field Ising model taken in various bases. We show also that TetrisCNN can detect more complex order parameters using the example of two-dimensional Ising gauge theory. This work can lead to the integration of NNs with quantum simulators to study new exotic phases of matter.

1 Introduction

Machine learning promises a revolution in quantum sciences [1–4], similar to the current revolution in industry [5]. Recently, neural networks (NNs) have become a powerful tool for detecting phases of matter [6–13], which is especially promising in the context of experimental data [14–19]. An ultimate goal of this forefront is to make such automated approaches interpretable [20–28], which
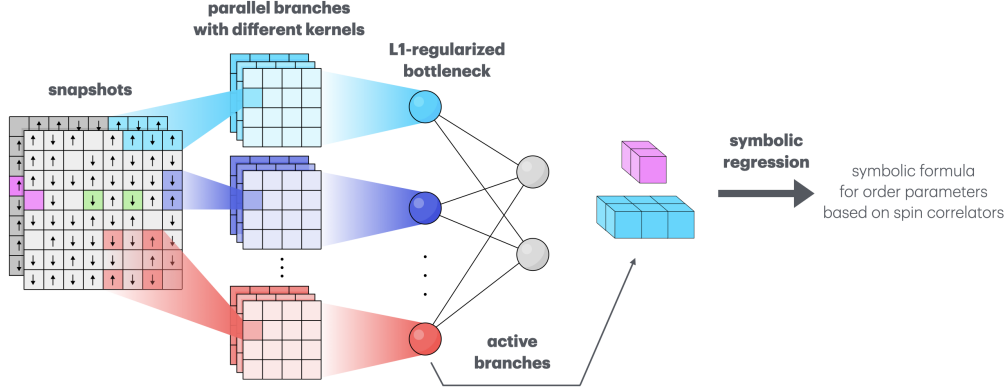
---

[*]Corresponding author.

Figure 1: **TetrisCNN is composed of parallel branches that use kernels of different shapes that map to spin correlators.** Its regularized bottleneck suppresses activations of branches containing non-unique or irrelevant information. The remaining active branches are analyzed with symbolic regression (SR), which provide a symbolic formula for the network based on spin correlators.

should lead to learning phases of matter and understanding their properties, in particular the detected order parameters [18, 19, 29–39].

Here, we report on designing a convolutional neural network (CNN) with parallel convolutional branches that use kernels of different sizes and shapes. We dub this network 'TetrisCNN' and show here that it detects the phases of spin systems and expresses their order parameters in a symbolic form based on spin correlators. TetrisCNN takes inspiration from Refs. [29, 36] and improves on them in terms of simplicity, computational cost, and inclusion of multiple measurement bases. Moreover, we use symbolic regression (SR) [40] to provide symbolic formulas for the detected order parameters and the network itself, making our method particularly attractive to physicists.

2   Methods

We test TetrisCNN ability to detect phase transitions and relevant order parameters on the example of three datasets derived from two paradigmatic spin models: 1D quantum transverse-field Ising model (TFIM) and 2D Ising gauge theory (IGT). Each spin at site i can take only two values $S_i = \pm 1$. We use $\overline{S_i} = \frac{1}{N} \sum_i S_i$ to symbolize averaging over all N system sites.

1D transverse-field Ising model (TFIM).   1D TFIM describes a spin chain of length N that exhibits quantum phases. The Hamiltonian of the system is given by:

$$\hat{H}_{\text{TFIM}} = -J\left(\sum_i \hat{S}_i^z \hat{S}_{i+1}^z + g \sum_i \hat{S}_i^x\right) \tag{1}$$

where J is the spin-spin interaction strength, g is the transverse field strength, and $\hat{S}^z$ and $\hat{S}^x$ are Pauli matrices representing the spin operators. When J dominates g, the system is in the ordered ferromagnetic phase with all spins aligned with each other in the z direction. The critical point of the system occurs when g balances J, and the system transitions to the disordered (paramagnetic) phase. We obtain ground states of 1D TFIM for various values of g ($J = 1$, $N = 150$) using the Density Matrix Renormalization Group (DMRG) [41] and ITensors.jl [42]. We take measurements (snapshots) of ground states in z and y bases and use them as input data for TetrisCNN to imitate experimental measurements on a quantum system.

2D Ising gauge theory (IGT) model   IGT is a classical spin model that exhibits a topological phase of matter, defined on a square lattice with periodic boundary conditions. This time the interaction between spins is defined within plaquettes p on the lattice and is of four-body nature:

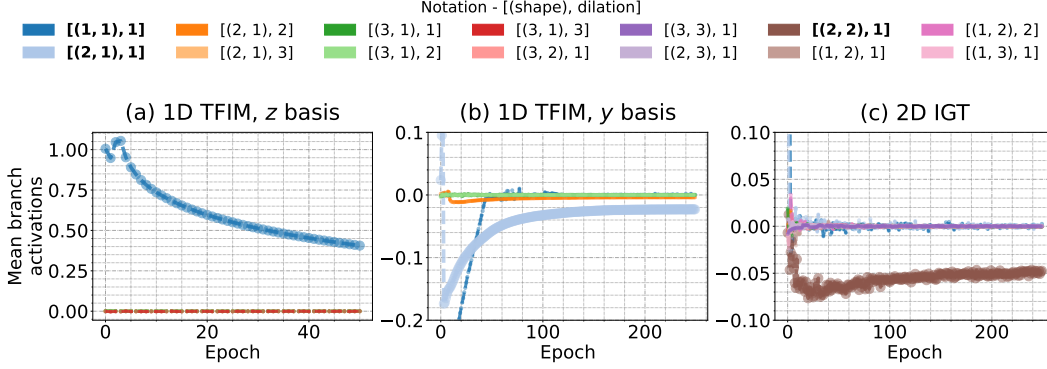$$H_{\text{IGT}} = -J \sum_p \prod_{i \in p} S_i \,. \tag{2}$$

Figure 2: **Activation of TetrisCNN branches across training on different datasets**. During training, branches without uniquely important information die out. In all three datasets, only one branch remains active. It relies on (a) the (1,1) kernel in the 1D TFIM measured in z basis, (b) the (2,1) kernel in the 1D TFIM measured in y basis, and (c) (2,2) kernel in the 2D IGT.

The (highly degenerate) ground state of this Hamiltonian meets the local constraint that the product of spins along each plaquette is $\prod_{i \in p} S_i = 1$. The system exhibits a transition from the low-temperature topological phase to the high-temperature phase with violated constraints (see Sec. 3.1.2 in Ref. [3]). We obtain spin configurations of IGT for different temperatures using the Monte Carlo method [9].

Architecture of TetrisCNN   The main idea behind TetrisCNN is to allow the network to utilize a multitude of differently shaped convolutional kernels, similar to Tetris pieces, within multiple parallel branches, as presented in Fig. 1. The results of their parallel computation, after global average pooling, enter the TetrisCNN bottleneck as a_k, which we regularize with $L_{\text{bottle}} = \sum_k \lambda_k |a_k|$, where k is a branch index. As such, we deactivate branches without important and unique information. We additionally choose λ_k so that they promote the use of simpler kernels over more complicated ones. After the bottleneck, fully-connected layers solve the posed task, given the learned sparse data representation. We describe the architecture in more detail in App. A.

TetrisCNN interpretation   If we apply TetrisCNN now to spin configurations (where each input element can be only $S_i = \pm 1$), due to TetrisCNN construction, each branch activation a_k can be mapped exactly to a linear function of a specific spin correlator (see App. A.1 and Ref. [29] for more details). For example, the branch using kernel (2,1) can only learn a nearest-neighbor spin correlator $\overline{S_i S_{i+1}}$ (it can also learn a one-body $\overline{S_i}$ but we can ignore it if the branch using (1,1) got deactivated). Therefore, we can easily understand what each branch computes, and linear regression is enough to find a formula for a_k. The bottleneck sparsity also enables obtaining a symbolic formula for the whole network using SR [40] (see App. A.2), which otherwise fails due to the large input size. We stress that the SR analysis is a 'cherry on top' (hopefully making it easier for physicists to engage with this work) because the analysis of the branch activations is enough to detect dominant spin correlators in the data.

Task   In the following, we combine TetrisCNN with the prediction-based method [43, 9], which allows for an unsupervised detection of phase transitions. First, the network is trained in a regression task to predict a tuning parameter of the system. For 1D TFIM, it is the transverse field value, g; for 2D IGT, it is an inverse temperature, β. Then, the phase transition is identified by locating the maximum of the derivative of the network output as a function of the label. Intuitively, the prediction-based method relies on the difficulty of a successful regression in the transition vicinity. We provide the implementation details in App. B and in the accompanying GitHub repository [44].

3   Results

The branch activation depends on relevant correlators present in the data   The interpretability of TetrisCNN relies on the sparsity of its bottleneck, i.e., a small number of its non-zero elements.
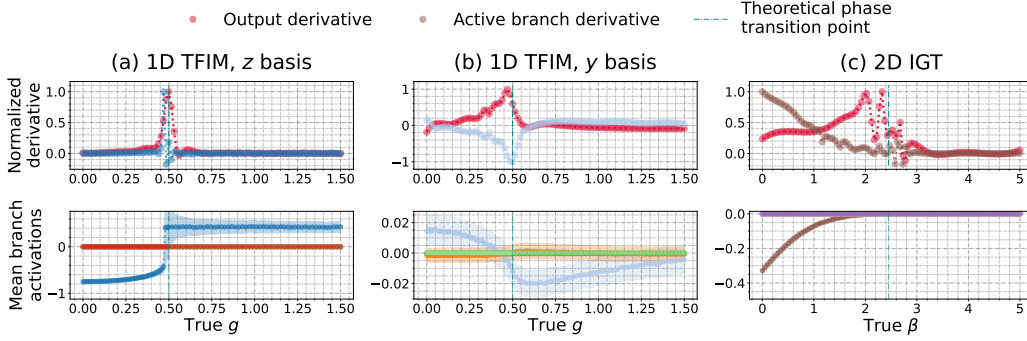
Figure 3: **Identification of phase transition location and the relevant spin correlators with TetrisCNN and prediction-based method.** (a)-(b) Results for 1D TFIM measures in z and y basis, respectively. (c) Results for 2D IGT.

Ideally, the bottleneck should contain only crucial information on the dataset. In Fig. 2, we show how, during training, TetrisCNN branches that pick up irrelevant correlations in the data deactivate, i.e., their respective activations in the bottleneck go to zero. The branch (or branches) that remains active uses the kernel that computes the relevant correlators present in the data.

For 1D TFIM, the order parameter that determines the transition between the ferro- and paramagnetic phases is the system magnetization in the z direction, $\langle \widehat{S}_i^z \rangle$. It is a one-body quantity, and as expected, in Fig. 2(a) we see that the remaining active branch of TetrisCNN uses the kernel of shape (1,1). However, the magnetization in the y direction does not have information on the phase transition. In Fig. 2(b) we see that when TetrisCNN analyzes TFIM measurements taken in y basis, it uses the branch with the kernel of shape (2,1). Indeed, we can check that the value of the nearest-neighbor spin correlator $\langle \widehat{S}_i^y \widehat{S}_{i+1}^y \rangle$ changes fast when the system undergoes a phase transition. Finally, TetrisCNN correctly focuses on the relevant correlator of the IGT, i.e., a four-body spin correlator related to the plaquette. We study how the detected correlators depend on λ_k in L_{bottle} in App. C.

TetrisCNN identifies the phase transition location in an unsupervised way In the first row of Fig. 3, we recover unsupervised detection of phases from Refs. [43, 9]. The predicted transition locations are in good agreement with the theoretical predictions for both Ising models. In the second row of Fig. 3, we plot the branch activations as functions of the respective label, i.e., g for 1D TFIM and β for 2D IGT. We see that the values of the active branches exhibit a fast change in the vicinity of the phase transition, reminding of the expected behavior of the order parameter. As a result, we can also recover the transition location by studying the derivative of the branch activation for data for different labels. The maxima of the derivatives of the branch activations and of the network output are slightly shifted, indicating different locations for the phase transition. The reason for this will become clear in the next section.

TetrisCNN identifies the order parameter and provides its symbolic formula Most excitingly, we can analyze the trained TetrisCNN with SR. For brevity, we focus here on results for TetrisCNN trained on 1D TFIM in y basis and place the rest in App. D. The first mapping we find is between the dominant branch activation and the spin correlation it detects. As seen in Fig. 2(b), TetrisCNN has an active branch that uses [(2,1),1] kernel and its activation is

$$a_{[(2,1),1]} = 0.122 \langle \overline{S_i^y S_{i+1}^y} \rangle + 0.0015 \; {\scriptstyle (R^2=1.00)} \,, \tag{3}$$

where $\langle \cdot \rangle$ means averaging over snapshots or spin configurations obtained for the same tuning parameter. R^2 coefficient in parenthesis shows that the fit is excellent. We see that in 1D TFIM, the active branches of TetrisCNN learn the magnetization in the z direction from the data measured in the z basis and a nearest-neighbor spin correlator in y direction from the data measured in the y basis. In 2D IGT, TetrisCNN branch is a linear function of the plaquette $\langle \overline{S_{(i,j)}^1 S_{(i,j+1)}^1 S_{(i,j+1)}^2 S_{(i-1,j+1)}^2} \rangle$.

4

The next step is to find the mapping between the non-zero branch activations and the network output. In the same example, the predicted g is

$$g = 107.985 \left(\left(\left(\overline{\langle S_i^y S_{i+1}^y \rangle} + 0.16 \right)^2 - 0.067 \right)^2 + 0.226 \right) \; {\scriptstyle (R^2 = 1.00)} . \tag{4}$$

As we see, the full network is a cumulant of the detected spin correlator. This explains the shift in the phase transition location predicted from the derivative of the branch activation compared to the derivative of the network output. We leave the question of which is a better indicator of the phase transition for future work.

4 Limitations and Outlook

An obvious limitation of TetrisCNN is the combinatorial complexity of kernels that increases fast for highly non-local order parameters. Currently, it can handle correlators up to (5,5). However, the main challenge is at the level of finding a symbolic formula for the whole network, while interpreting the bottleneck scales much better thanks to the direct linear mapping between the kernel shapes and spin correlators. In a way, TetrisCNN automatically searches for relevant correlators [45], but also learns an arbitrary useful function of correlators. Moreover, the learned correlators should be task-dependent, but we leave this aspect for future work.

The next steps are to expand TetrisCNN to spin models on lattices of different geometries. An exciting avenue is to modify the approach to detect and give formulas for long-range, nematic, and topological orders in data, following the developments of Refs. [19, 39]. Although TetrisCNN was developed with spin models in mind, it may tackle other correlation-based tasks, e.g., in molecular gases.

References

[1] J. Carrasquilla, *Machine learning for quantum matter*, Adv. Phys.: X **5**, 1797528 (2020).

[2] M. Krenn, J. Landgraf, T. Foesel, and F. Marquardt, *Artificial intelligence and machine learning for quantum technologies*, Phys. Rev. A **107**, 010101 (2023).

[3] A. Dawid, J. Arnold, B. Requena, A. Gresch, M. Płodzień, K. Donatella, K. A. Nicoli, P. Stornati, R. Koch, M. Büttner, R. Okuła, G. Muñoz-Gil, R. A. Vargas-Hernández, A. Cervera-Lierta, J. Carrasquilla, V. Dunjko, M. Gabrié, P. Huembeli, E. van Nieuwenburg, F. Vicentini, L. Wang, S. J. Wetzel, G. Carleo, E. Greplová, R. Krems, F. Marquardt, M. Tomza, M. Lewenstein, and A. Dauphin, Modern applications of machine learning in quantum sciences (2023), arXiv:2204.04198 [quant-ph] .

[4] M. Medvidović and J. R. Moreno, *Neural-network quantum states for many-body physics*, Eur. Phys. J. Plus **139**, 631 (2024).

[5] A. Dawid and Y. LeCun, *Introduction to latent variable energy-based models: a path toward autonomous machine intelligence*, J. Stat. Mech. **2024**, 104011 (2024).

[6] J. Carrasquilla and R. G. Melko, *Machine learning phases of matter*, Nat. Phys. **13**, 431 (2017).

[7] E. P. L. Van Nieuwenburg, Y.-H. Liu, and S. D. Huber, *Learning phase transitions by confusion*, Nat. Phys. **13**, 435 (2017).

[8] S. J. Wetzel, *Unsupervised learning of phase transitions: From principal component analysis to variational autoencoders*, Phys. Rev. E **96**, 022140 (2017).

[9] E. Greplova, A. Valenti, G. Boschung, F. Schäfer, N. Lörch, and S. D. Huber, *Unsupervised identification of topological phase transitions using predictive models*, New J. Phys. **22**, 045003 (2020).

[10] A. Bohrdt, S. Kim, A. Lukin, M. Rispoli, R. Schittko, M. Knap, M. Greiner, and J. Léonard, *Analyzing nonequilibrium quantum states through snapshots with artificial neural networks*, Phys. Rev. Lett. **127**, 150504 (2021).

[11] Z. Patel, E. Merali, and S. J. Wetzel, *Unsupervised learning of rydberg atom array phase diagram with siamese neural networks*, New J. Phys. **24**, 113021 (2022).

[12] J. Arnold, F. Schäfer, A. Edelman, and C. Bruder, *Mapping out phase diagrams with generative classifiers*, Phys. Rev. Lett. **132**, 207301 (2024).

[13] H. Kim, Y. Zhou, Y. Xu, K. Varma, A. H. Karamlou, I. T. Rosen, J. C. Hoke, C. Wan, J. P. Zhou, W. D. Oliver, Y. D. Lensky, K. Q. Weinberger, and E.-A. Kim, Attention to quantum complexity (2024), arXiv:2405.11632 [quant-ph] .

[14] B. S. Rem, N. Käming, M. Tarnowski, L. Asteria, N. Fläschner, C. Becker, K. Sengstock, and C. Weitenberg, *Identifying quantum phase transitions using artificial neural networks on experimental data*, Nat. Phys. **15**, 917 (2019).

[15] E. Khatami, E. Guardado-Sanchez, B. M. Spar, J. F. Carrasquilla, W. S. Bakr, and R. T. Scalettar, *Visualizing strange metallic correlations in the two-dimensional fermi-hubbard model with artificial intelligence*, Phys. Rev. A **102**, 033326 (2020).

[16] N. Käming, A. Dawid, K. Kottmann, M. Lewenstein, K. Sengstock, A. Dauphin, and C. Weitenberg, *Unsupervised machine learning of topological phase transitions from experimental data*, Mach. Learn.: Sci. Technol. **2**, 035037 (2021).

[17] M. Link, K. Gao, A. Kell, M. Breyer, D. Eberz, B. Rauf, and M. Köhl, *Machine learning the phase diagram of a strongly interacting fermi gas*, Phys. Rev. Lett. **130**, 203401 (2023).

[18] C. Miles, R. Samajdar, S. Ebadi, T. T. Wang, H. Pichler, S. Sachdev, M. D. Lukin, M. Greiner, K. Q. Weinberger, and E.-A. Kim, *Machine learning discovery of new phases in programmable quantum simulator snapshots*, Phys. Rev. Res. **5**, 013026 (2023).

[19] N. Sadoune, I. Pogorelov, C. L. Edmunds, G. Giudici, G. Giudice, C. D. Marciniak, M. Ringbauer, T. Monz, and L. Pollet, Learning symmetry-protected topological order from trapped-ion experiments (2024), arXiv:2408.05017 [quant-ph] .

[20] A. Dawid, P. Huembeli, M. Tomza, M. Lewenstein, and A. Dauphin, *Phase detection with neural networks: interpreting the black box*, New J. Phys. **22**, 115001 (2020).

[21] S. J. Wetzel, R. G. Melko, J. Scott, M. Panju, and V. Ganesh, *Discovering symmetry invariants and conserved quantities by interpreting siamese neural networks*, Phys. Rev. Res. **2**, 033499 (2020).

[22] A. Dawid, P. Huembeli, M. Tomza, M. Lewenstein, and A. Dauphin, *Hessian-based toolbox for reliable and interpretable machine learning in physics*, Mach. Learn.: Sci. Techn. **3**, 015002 (2021).

[23] J. Arnold, F. Schäfer, M. Žonda, and A. U. J. Lode, *Interpretable and unsupervised phase classification*, Phys. Rev. Research **3**, 033052 (2021).

[24] J. Arnold and F. Schäfer, *Replacing neural networks by optimal analytical predictors for the detection of phase transitions*, Phys. Rev. X **12**, 031044 (2022).

[25] J. Arnold, N. Lörch, F. Holtorf, and F. Schäfer, Machine learning phase transitions: Connections to the fisher information (2023), arXiv:2311.10710 [cond-mat.dis-nn] .

[26] S. J. Wetzel, Closed-form interpretation of neural network classifiers with symbolic regression gradients (2024), arXiv:2401.04978 [cs.LG] .

[27] Y. Zhan, A. Elben, H.-Y. Huang, and Y. Tong, *Learning conservation laws in unknown quantum dynamics*, PRX Quantum **5**, 010350 (2024).

[28] K. Cybinski, M. Płodzień, M. Tomza, M. Lewenstein, A. Dauphin, and A. Dawid, *Characterizing out-of-distribution generalization of neural networks: application to the disordered Su-Schrieffer-Heeger model*, Mach. Learn.: Sci. Technol. 10.1088/2632-2153/ad9079 (2024).

[29] S. J. Wetzel and M. Scherzer, *Machine learning of explicit order parameters: From the Ising model to SU(2) lattice gauge theory*, Phys. Rev. B **96**, 184410 (2017).

[30] J. Greitemann, K. Liu, and L. Pollet, *Probing hidden spin order with interpretable machine learning*, Phys. Rev. B **99**, 060404 (2019).

[31] K. Liu, J. Greitemann, and L. Pollet, *Learning multiple order parameters with interpretable machines*, Phys. Rev. B **99**, 104410 (2019).

[32] J. Greitemann, K. Liu, L. D. C. Jaubert, H. Yan, N. Shannon, and L. Pollet, *Identification of emergent constraints and hidden order in frustrated magnets using tensorial kernel methods of machine learning*, Phys. Rev. B **100**, 174408 (2019).

[33] N. Sadoune, G. Giudici, K. Liu, and L. Pollet, *Unsupervised interpretable learning of phases from many-qubit systems*, Phys. Rev. Res. **5**, 013082 (2023).

[34] N. Sadoune, K. Liu, H. Yan, L. D. C. Jaubert, N. Shannon, and L. Pollet, Human-machine collaboration: ordering mechanism of rank-2 spin liquid on breathing pyrochlore lattice (2024), arXiv:2402.10658 [cond-mat.str-el] .

[35] A. Cole, G. J. Loges, and G. Shiu, *Quantitative and interpretable order parameters for phase transitions from persistent homology*, Phys. Rev. B **104**, 104426 (2021).

[36] C. Miles, A. Bohrdt, R. Wu, C. Chiu, M. Xu, G. Ji, M. Greiner, K. Q. Weinberger, E. Demler, and E.-A. Kim, *Correlator convolutional neural networks as an interpretable architecture for image-like quantum matter data*, Nat. Commun. **12**, 3905 (2021).

[37] H. Schlömer and A. Bohrdt, *Fluctuation based interpretable analysis scheme for quantum many-body snapshots*, SciPost Phys. **15**, 099 (2023).

[38] C. Cao, F. M. Gambetta, A. Montanaro, and R. A. Santos, Unveiling quantum phase transitions from traps in variational quantum algorithms (2024), arXiv:2405.08441 [quant-ph] .

[39] A. Suresh, H. Schlömer, B. Hashemi, and A. Bohrdt, Interpretable correlator transformer for image-like quantum matter data (2024), arXiv:2407.21502 [cond-mat.quant-gas] .

[40] M. Cranmer, Interpretable machine learning for science with PySR and SymbolicRegression.jl (2023), arXiv:2305.01582 [astro-ph.IM] .

[41] S. R. White, *Density matrix formulation for quantum renormalization groups*, Phys. Rev. Lett. **69**, 2863 (1992).

[42] E. Stoudenmire and D. J. Schwab, *Supervised learning with tensor networks*, in *Advances in Neural Information Processing Systems*, Vol. 29, edited by D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett (Curran Associates, Inc., 2016).

[43] F. Schäfer and N. Lörch, *Vector field divergence of predictive model output as indication of phase transitions*, Phys. Rev. E **99**, 062107 (2019).

[44] K. Cybiński, J. Enouen, A. Georges, and A. Dawid, GitHub repository with the demonstration code for this paper (2024), doi:10.5281/zenodo.14035852.

[45] R. Verdel, V. Vitale, R. K. Panda, E. D. Donkor, A. Rodriguez, S. Lannig, Y. Deller, H. Strobel, M. K. Oberthaler, and M. Dalmonte, *Data-driven discovery of statistically relevant information in quantum simulators*, Phys. Rev. B **109**, 075152 (2024).

[46] T.-Y. Lin, A. RoyChowdhury, and S. Maji, *Bilinear CNN models for fine-grained visual recognition*, in *Proceedings of the IEEE international conference on computer vision* (2015) pp. 1449–1457.

[47] J. Ansel, E. Yang, H. He, N. Gimelshein, A. Jain, M. Voznesensky, B. Bao, P. Bell, D. Berard, E. Burovski, G. Chauhan, A. Chourdia, W. Constable, A. Desmaison, Z. DeVito, E. Ellison, W. Feng, J. Gong, M. Gschwind, B. Hirsh, S. Huang, K. Kalambarkar, L. Kirsch, M. Lazos, M. Lezcano, Y. Liang, J. Liang, Y. Lu, C. K. Luk, B. Maher, Y. Pan, C. Puhrsch, M. Reso, M. Saroufim, M. Y. Siraichi, H. Suk, S. Zhang, M. Suo, P. Tillet, X. Zhao, E. Wang, K. Zhou, R. Zou, X. Wang, A. Mathews, W. Wen, G. Chanan, P. Wu, and S. Chintala, *Pytorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation*, in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS '24 (Association for Computing Machinery, New York, NY, USA, 2024) p. 929–947.
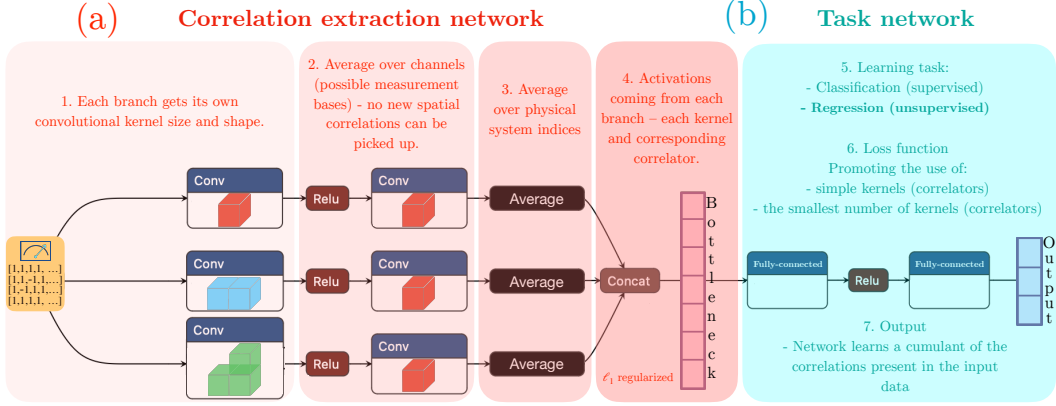
Figure 4: **A detailed presentation of TetrisCNN architecture and how it processes spin configurations at each step.** (a) The first part of TetrisCNN, the correlation extraction network, which maps the spin configurations onto functions of many-body spin correlators present in the data. (b) The second TetrisCNN subnetwork is the task network, in our case a fully-connected network. If applied to phase detection problem, the task can bebe either unsupervised (as here, where we use the prediction-based method) or supervised.

A Detailed explanation of the TetrisCNN architecture

A.1 TetrisCNN architecture

The TetrisCNN architecture can be divided into two general parts: correlation extraction and task networks. They are presented in Fig. 4(a)-(b), respectively, along with a detailed description of each data processing step. We discuss these steps in the following sections.

Extracting correlations. Let us first focus on the correlation extraction network. The input to the network are either classical spin configurations or 'snapshots' - projective measurements on a simulated or experimentally realized quantum spin lattice. The architecture supports either one- or two-dimensional systems with snapshots in any of x, y, z measurement bases or their combination. Each combination of input bases results in a separate training and analysis routines. The snapshots are introduced into the network and then processed in parallel by several 'branches', reminiscent of bilinear CNNs from Ref. [46]. Each branch consists of a convolutional layer and then averages over the number of filters and physical indices, thus reducing each snapshot to a single number - a branch activation a_k. Following the logic presented by Ref. [8], we interpret this activation as a function of all correlators present in the data of the same shape and size or smaller as the convolutional kernel used in the branch. Therefore, each activation a_k can be written as a function of the input snapshot - set of spin variables S_i:

$$a_k = f(\{S_i^z\}) \tag{5}$$

We map the spin variables from $\{\uparrow, \downarrow\}$ to $\{-1, 1\}$. Then, further following the logic from Ref. [29], we leverage the fact that:

$$(S_i^d)^p = \begin{cases} 1 & \text{if } p \text{ is even} \\ S_i^d & \text{if } p \text{ is odd} \end{cases}, \tag{6}$$

and apply it to a Taylor expansion of the function:

$$f(S_i^z) = f_0 + f_1 S_i^z + f_1 \underbrace{(S_i^z)^2}_{1} + f_2 \underbrace{(S_i^z)^3}_{S_i^z} + \dots \tag{7}$$
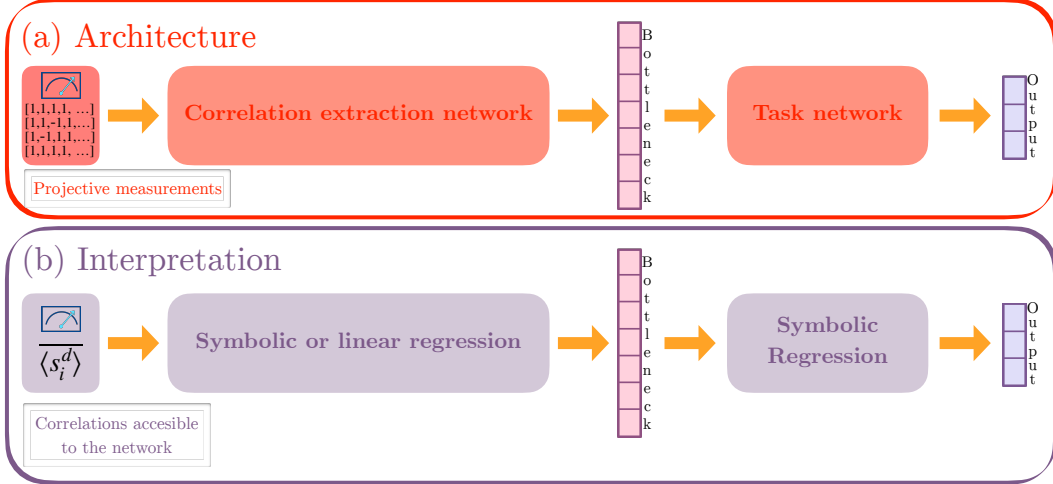
8

Figure 5: **A high-level overview of the TetrisCNN architecture and its interpretation.** The two parts form a pipeline for the unsupervised detection of phases of matter and their order parameters. (a) TetrisCNN architecture comprises two general parts - correlation extraction network and task network. The two are joined by a bottleneck, which is key to the subsequent analysis. (b) Obtaining symbolic formulas takes two steps. First, finding a mapping between correlations accessible to each branch and this branch activation with linear or symbolic regression. Second, finding a mapping between the network output and the branch activations in the bottleneck.

This leaves us with an activation a_k being a function of all correlations of smaller or equal size than the kernel size, but only of the first order. An example of this for kernels (1,1) and (2,1) would be:

$$f\left(s_i^z\right) = F\left(\frac{1}{N}\sum_i S_i^z\right) \tag{8}$$

$$f\left(s_i^y, s_{i+1}^y\right) = F\left(\frac{1}{N}\sum_i S_i^y\right) + F\left(\frac{1}{N-1}\sum_i S_i^y S_{i+1}^y\right) \tag{9}$$

Truncation of search space. This trick still leaves us with the branch activation dependent on the correlations also smaller than the kernel size. To tackle this, we add a term to the loss function so that the network learns during the training to use only the simplest non-trivial kernels to achieve this task. This is achieved by combining the activations from all branches, forming a network bottleneck. This bottleneck is ℓ_1 regularized, and the values of the activations are combined into L_{bottle}, which is then added to the final loss function. As a result, once the network is trained, we can inspect the values of the activations in the analysis routine, and the ones with the highest amplitude are perceived as the most important to the network. Therefore, e.g., if we allow the network to be used in the optimization process, the kernels (1,1), (2,1), (3,1), and kernel (2,1) turns out to be the one with the highest amplitude, the function describing it presented in Eq. (9) collapses only to the second term. The first term is either zero or not of the leading order because if it were of the leading order, then a kernel (1,1) would not be deactivated. Therefore, only by studying the branch activations, we can detect leading correlations in the data, solely from the interpretable design of TetrisCNN architecture.

Learning task. The second part of the network is the task network, which learns the mapping between the branch activations and the desired output. We implement two possible tasks: classification and regression. The variant with regression task is based on the prediction-based method [43, 9], which allows for an unsupervised detection of the phase transition point, as opposed to the classification task, which requires the user to supply the labels.

A.2 Interpretation with symbolic regression

Although the detection of relevant spin correlators can be done simply by studying the network bottleneck and leveraging its interpretable design, we can additionally use regression techniques to

provide a symbolic formula for both bottleneck activations and the network output, all in terms of spin correlators. To this end, we first find a mapping between the spin correlators and the branch activations. Due to ℓ_1 regularization of the bottleneck, increasing λ_k, and binary values of the spin data, the bottleneck elements a_k can only be linear functions of the respective spin correlators, as explained above (i.e., kernel (1,1) can only learn $\overline{S_i}$, kernel (2,1) - only $\overline{S_i S_{i+1}}$). We can identify this mapping using linear regression (e.g., via least-squares minimization). However, to find a symbolic formula for the mapping between bottleneck activations a_k and network output, we need to use symbolic regression (SR). These two steps are presented in Fig. 5(b).

Symbolic regression. Symbolic regression (SR) is a type of regression analysis that searches for mathematical expressions that best fit the given data rather than assuming a predefined functional form like in traditional regression. This approach is particularly useful in physics, where the goal is often to discover underlying equations or relationships that govern the observed data [40]. The SR analysis was performed using the PySR package, a Python interface to the SymbolicRegression.jl library in Julia [40]. This package employs evolutionary algorithms and genetic programming to iteratively search for and evolve mathematical expressions that best fit the activation data. Its procedure is the following:

1. **Initialization:** The algorithm starts with a population of random mathematical expressions involving basic operators ($+$, $-$, $*$, $/$), constants, and the input variables (correlators).

2. **Evaluation:** Each expression is evaluated based on its ability to predict the kernel activations. The evaluation metric is the default SymbolicRegression.jl loss function, which is the mean squared error (MSE) between the predicted and actual activations.

3. **Selection and Evolution:** The top-performing expressions are selected, and new expressions are generated through genetic operations such as mutation (altering parts of an expression) and crossover (combining parts of two expressions).

4. **Convergence:** Depending on the size of the input, the process continues for 6000 or 18000 epochs, which allows for a good convergence of each branch evolved by the genetic algorithm employed under the hood of PySR.

B Details on the numerical implementation and used hyperparameters

Task In this work, TetrisCNN is used in combination with the prediction-based method [43, 9]. We train TetrisCNN using PyTorch [47] by minimizing the mean squared error between the network output and the tuning parameter. For 1D TFIM, the tuning parameter is the transverse field value, g; for 2D IGT, the tuning parameter is an inverse temperature, β. The training hyperparameters are in Tab. 1.

Input data The input data are spin configurations, either classical (2D IGT) or projective measurements taken on a quantum system in some bases (1D TFI). Single input data therefore is a vector (for 1D systems) or matrix of $\{-1, 1\}$ (for 2D systems). If a system is classical or only one measurement basis is considered, input data has a single channel, as in case of the 1D TFI, where z and y bases were considered separately. If two measurement bases are considered, like in the 2D IGT case, the spin configurations (here from two sublattices) are randomly bunched into pairs consisting of two bases, and are fed into two input channels. For three measurement bases, there are three channels.

Declaring available branches TetrisCNN requires giving a list of kernels that it needs to consider. The kernel format we use is $[(\mathrm{dimension}_1, \mathrm{dimension}_2), \mathrm{dilation}]$, where $\mathrm{dimension}_2 = 1$ for 1D and dilation inserting holes of specified size between the kernel consecutive elements. Dilation $= 1$ means no hole, dilation $= 2$ means hole of one element size. Each kernel is then used by a separate parallel convolutional branch. In our numerical experiments, we used up to 20 parallel branches and did not notice any performance drop. We report the list of kernels that we made available for every setup described in the main text in Tab. 1.

Bottleneck regularization Interpretation of TetrisCNN via symbolic regression (SR) hinges upon the sparsity of its bottleneck. Therefore, we regularize the bottleneck elements a_k, i.e., branch

Table 1: Hyperparameters of TetrisCNN trained on three datasets, whose results are reported in the main text

	1D TFIM, z basis	1D TFIM, y basis	2D IGT
Learning rate	1e-2	5e-4	5e-2
Weight decay	1e-2	1e-1	1e-5
Optimizer	Adagrad	AdamW	Adagrad
Max epochs	100	100	1500
Early stopping	Yes	No	No
Normalized input	No	No	MaxMinScaler
Available kernels	[(1, 1), 1], [(2, 1), 1], [(2, 1), 2], [(2, 1), 3], [(3, 1), 1], [(3, 1), 2], [(3, 1), 3]	[(1, 1), 1], [(2, 1), 1], [(2, 1), 2], [(3, 1), 1], [(3, 1), 2]	[(1, 1), 1], [(2, 1), 1], [(1, 2), 1], [(2, 2), 1], [(2, 1), 2], [(1, 2), 2], [(3, 1), 1], [(3, 2), 1], [(1, 3), 1], [(2, 3), 1], [(3, 3), 1]
Bottleneck size (# branches)	7	5	11
$\lambda_{\min}, \lambda_{\max}$	1e-4, 1e0	1e-4, 1e-1	1e-5, 1e-1
Task network	[# branches, 32, 1]	[# branches, 32, 16, 1]	[# branches, 32, 16, 1]

activations averaged across channels and physical system size via the following loss term:

$$L_{\text{bottle}} = \lambda_k \sum_k |a_k| \,. \tag{10}$$

This ℓ_1 regularization encourages the bottleneck elements to go to zero, which translates into the use of the smallest number of branches (therefore kernels and spin correlators) possible. We additionally modulate the values of λ_k so it is the smallest for branches using kernels of the smallest size, also favoring simpler correlations over more complex ones. We choose it so that they are equally distanced on a logaritmic scale between $\lambda_{\min}$ and $\lambda_{\max}$, $\lambda_k = $ np.logscale($\lambda_{\min}$, $\lambda_{\max}$). We report values of $\lambda_{\min}$ and $\lambda_{\max}$ in Tab. 1.

Architecture of the task network   While the correlation extraction network is the same across setups, we vary the fully-connected task network between the datasets. We report its architecture also in Tab. 1, where the input to the task network is the bottleneck (therefore branch activations) and the next numbers indicate numbers of hidden neurons in the consecutive fully-connected layers.

C   Interplay between the accuracy and sparsity of TetrisCNN via the bottleneck regularization

Here we study how the detected spin correlators (and kernels used by the active branches of TetrisCNN) depend on penalties λ_k put on the branch activations that use kernels of increasing complexity. We set $\lambda_k = $ np.logscale($\lambda_{\min}$, $\lambda_{\max}$), where $\lambda_{\min}$ is fixed to 10^{-4} and 10^{-5} for 1D TFIM and 2D IGT, respectively. They are applied to the branch activations a_k in the order of appearance on the list of the respective 'available kernels' defined by a user (see Tab. 1). In Fig. 6, in the first row, we plot the average regression performance in terms of R^2 of 5 TetrisCNNs trained with increasing $\lambda_{\max}$ that is with increasing differences between applied λ_k that put increasing preference towards smaller and simpler kernels (i.e., kernels from the beginning of the user-defined list). In the second row, we plot average branch activations of the same TetrisCNNs, which we normalize to the largest branch activation per network instance. We show here which kernels are preferred by differently regularized networks and also how stable is their choice across 5 initializations.

Uniform λ_k   When ℓ_1 penalties put on the branch activations corrresponding to kernels of increasing complexity are equal to each other, the regularization results only in a sparse bottleneck and in a
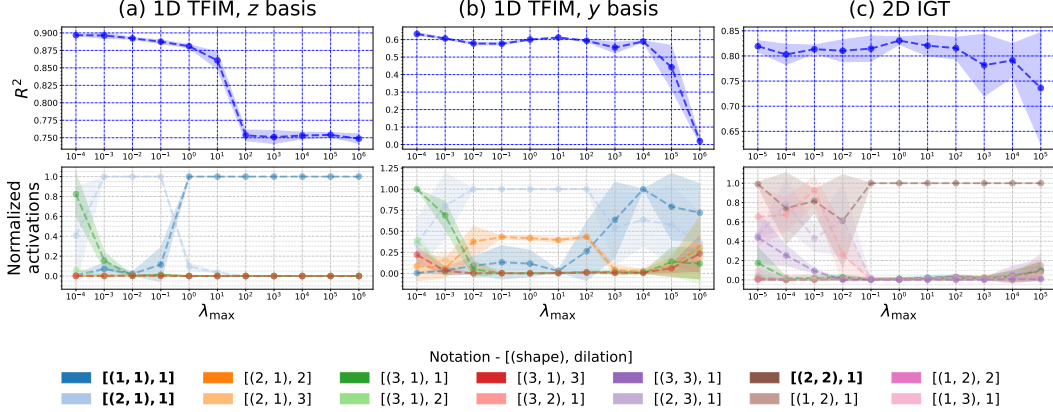
Figure 6: **Penalties of branch activations λ_k and the use of kernels.** We plot here the average R^2 of the network output (compared to the true label) and the normalized branch activations (where 1 is the largest branch activation value) of 5 TetrisCNN instances trained with different λ_{max}. The larger λ_{max}, the larger differences between penalties λ_k put on the branch activations that use kernels of increasing complexity. Increasing λ_{max} results in a sparser bottleneck of TetrisCNN that uses simpler and smaller kernels. At some critical λ_{max}, we can also get a significant R^2 drop, so the sparsity may come at the cost of performance.

network preference to use a single kernel. There is no preference towards simpler kernels yet. We see that for 1D TFIM in z basis, TetrisCNN tends to use kernel (3,1) with dilatation 1 accompanied by kernel (2,1) with dilatation 1. For 1D TFIM in y basis, the variety of used kernels is even larger, the dominant one also being kernel (3,1) with dilatation 1, but accompanied by kernels (3,1) with dilatation 2 and 3.

Non-uniform λ_k In the case of 1D TFIM in z basis, for a bit larger differences between λ_k ($\lambda_{max} \geq 10^{-3}$), the dominant kernel changes to (2,1), and then to kernel (1,1) for $\lambda_{max} > 10^{-1}$. These choices stay the same across 5 initializations. This kernel switch results only in a slight R^2 drop, suggesting that (1,1) learns a very relevant spin correlator. For y basis, starting from $\lambda_{max} = 10^{-2}$, all network instances use kernel (2,1). From $\lambda_{max} \geq 10^3$, the kernel choice becomes unstable and is accompanied by a large R^2 drop, suggesting that using smaller kernels is detrimental.

TetrisCNN trained on 2D IGT is less dependant on λ_k Finally, TetrisCNNs trained on 2D IGT samples, regardless of λ_k, usually use kernel (2,2) as the dominant kernel. For $\lambda_{max} \leq 10^{-2}$, the only difference is that the (2,2) kernel is accompanied by some larger kernels, and the choice becomes less stable across initializations. Interestingly, the largest R^2 is for intermediate $\lambda_{max} = 10^0$ suggesting that sparsity does not always come at the cost of performance drop.

The R^2 difference between TetrisCNNs trained on 1D TFIM snapshots from z and y basis Next to the discussion on λ_k and the use of kernels, the first row of Fig. 6(a)-(b) nicely suggests via the R^2 difference that the z basis contains better information for predicting the tuning parameter g of 1D TFIM than the y basis. Indeed, the g aims to align spins in the z direction, and it is the term breaking the symmetry underlying the phase transition. Following the regression performance between measurement bases is, next to the simplicity of the learned spin correlator, an important guidance on studying phases of matter with TetrisCNN using various measurements.

D   Symbolic regression results

The analysis process described in App. A.2 is presented here on the example of 1D TFIM model in y basis in Fig. 7. Panels (a) and (b) present the mapping between the spin correlators computed from the raw input data and the learned bottleneck activations as a function of the regression parameter. In all panels, the regression parameter is the transverse field strength g. In panel (b), we also show the least squares fit that includes all possible correlators from Eq. (9), i.e., $\langle \overline{S_i^y} \rangle$ and $\langle \overline{S_i^y S_{i+1}^y} \rangle$. The
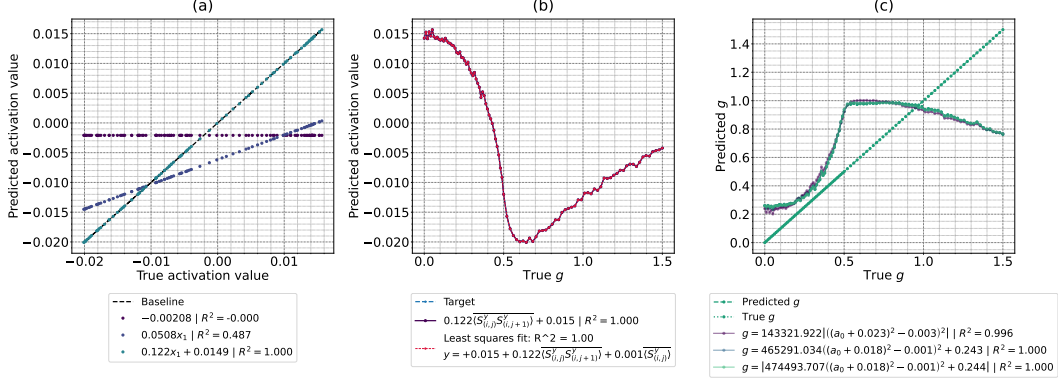
Figure 7: **Symbolic regression results for TetrisCNN trained on 1D TFIM measured in y basis.** (a) Fits for $a_{[(2,1),1]}(x_1)$, where $x_1 = \langle S_i^y S_{i+1}^y \rangle$. (b) Fits for $g(a_{[(2,1),1]})$. (c) Fits for $g(a_0)$, where $a_0 = \langle S_i^y S_{i+1}^y \rangle$ and comparison between the predicted g of the network based on snapshots from the y basis and the true g.

coefficient next to $\langle \overline{S_i^y S_{i+1}^y} \rangle$ is two orders of magnitude larger than the one next to $\langle \overline{S_i^y} \rangle$, indicating that the regularization of the bottleneck is working as planned. Finally, panel (c) presents the mapping between the bottleneck activations and the output of the network.

In Sec. 3 we present the distilled equation for TetrisCNN predicting transverse field strength g in the TFIM model out of measurements in y basis, but our analysis also spanned two other experimental settings: TFIM model in z basis and IGT model. In both cases, the symbolic distillation yielded an accurate description of the knowledge extracted by the network.

TFIM model in z basis. Here, the network correctly picked up the simplest nonzero correlator present in the data, which is the magnetization in the direction z, using kernel (1,1). The nonzero branch corresponding to this correlator is approximated in Eq. (11). Given this approximation, an equation that approximates the whole TetrisCNN is presented in Eq. (12).

$$a_{[(1,1),1]} = -1.176\langle \overline{S_i^z} \rangle + 0.426 \quad (R^2 = 1.00) \tag{11}$$

$$g = |2.798\langle \overline{S_i^z} \rangle - 1.013| \quad (R^2 = 0.97) \tag{12}$$

IGT model Analysis of data from IGT model presented us with a new insight into case previously studied by Ref. [9]. We knew from their investigation that the correlations picked up by the CNN must not exceed the receptive field of convolutional kernel (3,3), and using our analysis routine we were able to narrow this estimate from more than 250000 possibilities to just one meaningful correlation - a four body correlator within the convolutional kernel (2,2). Our analysis of the nonzero TetrisCNN branch revealed this kernel to be $\langle S_{(i,j)}^1 S_{(i,j+1)}^1 S_{(i,j+1)}^2 S_{(i-1,j+1)}^2 \rangle$, which is a product of all spins about a given vertex, producing a pattern similar to the vertex operator present in definition of the toric code (see Fig. 6 of Ref. [9]). Consecutive investigation of the raw data revealed that this correlator is indeed the only one in the (2,2) receptive field that varies continuously across the investigated β range. All other correlators of smaller or equal size converge to a constant value as the number of snapshots per β is increased. An equation describing the nonzero branch activation is presented in Eq. (13). The full distilled relation describing learned mapping from the input correlators to the regression parameter, inverse temperature β, is presented in Eqs. (14) - (15).

$$a_{[(2,2),1]} = 4.95\langle \overline{S_{(i,j)}^1 S_{(i,j+1)}^1 S_{(i,j+1)}^2 S_{(i-1,j+1)}^2} \rangle - 0.322 \; (R^2 = 0.99) \tag{13}$$

$$\beta = 1074.214(\langle \overline{S_{(i,j)}^1 S_{(i,j+1)}^1 S_{(i,j+1)}^2 S_{(i-1,j+1)}^2} \rangle + \tag{14}$$

$$+0.002|6421\overline{\langle S_{(i,j)}^1 S_{(i,j+1)}^1 S_{(i,j+1)}^2 S_{(i-1,j+1)}^2 \rangle} - 392| - 0.044)^2 + 0.015 \; (R^2 = 0.99) \tag{15}$$