Towards Effective Deep Learning for Constraint Satisfaction Problems

Hong Xu(✉), Sven Koenig, and T. K. Satish Kumar

University of Southern California, Los Angeles, CA 90089, United States of America
{hongx, skoenig}@usc.edu, tkskwork@gmail.com

Abstract. Many attempts have been made to apply machine learning techniques to constraint satisfaction problems (CSPs). However, none of them have made use of the recent advances in deep learning. In this paper, we apply deep learning to predict the satisfiabilities of CSPs. To the best of our knowledge, this is the first effective application of deep learning to CSPs that yields >99.99% prediction accuracy on random Boolean binary CSPs whose constraint tightnesses or constraint densities do not determine their satisfiabilities. We use a deep convolutional neural network on a matrix representation of CSPs. Since it is NP-hard to solve CSPs, labeled data required for training are in general costly to produce and are thus scarce. We address this issue using the asymptotic behavior of generalized Model A, a new random CSP generation model, along with domain adaptation and data augmentation techniques for CSPs. We demonstrate the effectiveness of our deep learning techniques using experiments on random Boolean binary CSPs. While these CSPs are known to be in P, we use them for a proof of concept.

1 Introduction

A lot of research has been dedicated to applying machine learning techniques to *constraint satisfaction problems* (CSPs), such as support vector machines [5], linear regression [27], decision tree learning [10,12], clustering [15,23], k-nearest neighbors [21], and so on [16]. However, there are a few drawbacks in these methods. First, they do not consistently produce extremely high (>99%) prediction accuracies. Secondly, to the best of our knowledge, they critically rely on handcrafted features. For different distributions of CSPs coming from different application domains and for different tasks of interest, the optimal features need to be carefully selected by humans accordingly [16]. How to select good features thus requires dedicated research [3,4].

Deep learning is a class of machine learning methods based on multi-layer (deep) *neural networks* (NNs). Thanks to the advent of "Big Data," it has significantly advanced during the past decade and achieved great success in many areas, such as computer vision and natural language processing [11]. In these applications, it consistently produces extremely high prediction accuracies, and often approaches or even surpasses human-level performance in many human perception tasks, such as object recognition [8, 14] and speech recognition [6].

Furthermore, it does not rely on handcrafted features. One of the key reasons for the success of deep learning is the availability of huge amounts of training data. However, due to the NP-hardness of CSPs, it is costly to label CSPs with properties such as their satisfiabilities and the best algorithms to solve them. This has become a roadblock for effective deep learning for CSPs. Indeed, a recent study shows that, without a huge amount of labeled data, a *convolutional NN* (cNN) for algorithm selection is ineffective for CSPs [19].[1]

In this work, we successfully apply deep learning to predict the satisfiabilities of random Boolean binary CSPs with high prediction accuracies (>99.99%). To the best of our knowledge, this is the first effective application of deep learning to random CSPs whose constraint tightnesses or constraint densities do not determine their satisfiabilities. Accurately predicting satisfiabilities might improve the dynamic variable ordering in a backtracking algorithm for CSPs to increase the likelihood of choosing a variable that results in a satisfiable subproblem so as to minimize backtracking. Further adapting our current method to qualitatively predict the number of solutions, e.g., "0," "1," and "≥1," might further improve the dynamic variable ordering. In addition, *transfer learning* may be potentially used to enable effective deep learning for other tasks such as predicting the most efficient algorithm and its best parameter settings for a given CSP.

In this paper, first, we describe the architecture of our cNN. Then, we address the issue of the lack of labeled data using the asymptotic behavior of *generalized Model A*, a new random CSP generation model, along with domain adaptation and data augmentation techniques for CSPs. We demonstrate the effectiveness of our techniques using experiments on random Boolean binary CSPs. While these CSPs are known to be in P, we use them as a proof of concept.

Preliminaries A CSP is formally defined as a tuple $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$, where $\mathcal{X} = \{X_1, \ldots, X_n\}$ is a set of variables, $\mathcal{D} = \{D_1, \ldots, D_n\}$ is a set of domains corresponding to their respective variables, and $\mathcal{C} = \{C_1, \ldots, C_m\}$ is a set of constraints. Each constraint $C_i \in \mathcal{C}$ is a pair $\langle S(C_i), R_i \rangle$, where $S(C_i)$ is a subset of $\mathcal{X}$ and R_i is a $|S(C_i)|$-ary relation that specifies incompatible and compatible assignments of values to variables in $S(C_i)$. In a *table constraint* C_i, R_i is a set of tuples, each of which indicates the compatibility of an assignment of values to variables in $S(C_i)$. A tuple is compatible if it specifies a compatible assignment of values to variables, and is otherwise incompatible. We focus on CSPs where all constraints are table constraints.

The concept of a cNN, a class of deep NN architectures, was initially proposed for an object recognition problem [18] and has recently achieved great success [8, 14]. It is a multi-layer feedforward NN that takes a multi-dimensional (usually 2-D or 3-D) matrix as input. While cNNs are mainly used for classification, they are also used for regression [24]. A cNN has three types of layers: *convolutional layers*, *pooling layers*, and *fully connected layers*. A convolutional layer performs a convolution operation. A pooling layer combines the outputs of several nodes

[1] Another related work [9] using a different approach was only publicly available after this paper was accepted, before which we had no access to it. Nevertheless, it only demonstrated low training and test accuracies in the experiments when the number of variables in a CSP is non-trivial (≥5) and we do not consider it effective (yet).

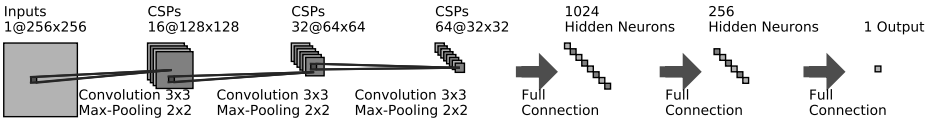

Fig. 1: The architecture of our CSP-cNN.

in the previous layer into a single node in the current layer. A fully connected layer connects every node in the current layer to every node in the previous layer.

2 Enabling Deep Learning for CSPs

In the context of deep learning for CSPs, each *data point* is a CSP (instance). If a data point is *labeled*, its label is a property of this CSP, such as its satisfiability, its K-consistency, the best algorithm to solve it, or the amount of time required to solve it with a specific algorithm. Our cNN takes a data point (a CSP) as input and predicts its label. In order to enable a cNN to take a CSP as input, we represent a binary CSP using a matrix as follows. Each row/column represents a variable-value pair (X_i, x_i). The element in (X_i, x_i)'s row and (X_j, x_j)'s column is zero if $\{X_i = x_i, X_j = x_j\}$ is disallowed; otherwise, the element is one. We refer to this matrix as a *CSP matrix*.

The rationale behind using a CSP matrix is the observation that it resembles a 2-D array of pixel values in a gray-scale image. cNNs are known to recognize patterns in a multi-dimensional array of numbers, such as patterns in an image in computer vision applications. Our intuition is that many properties of a CSP depend on the patterns of compatible and incompatible tuples in its constraints. Therefore, we expect a cNN that takes a CSP matrix as input to be able to recognize patterns of compatible and incompatible tuples to make its predictions.

Our cNN has the following architecture. Each node in the input layer corresponds to an element of the input CSP matrix. It has 4 convolutional layers with 3-by-3 kernels with stride 1, each of which is followed by a MaxPool layer with a 2-by-2 kernel with stride 2. We used "same" padding for all convolutional layers and "valid" padding for all MaxPool layers. Following these layers, there are 2 fully connected hidden layers. Finally, there is an output layer with a single node. The node in the output layer uses the *sigmoid activation function*, and all other neurons are *rectified linear units* (ReLUs) [13]. The output layer uses L2 regularization with a coefficient of 0.1, and all other layers use L2 regularization with a coefficient of 0.01. We refer to this cNN as CSP-cNN as shown in Fig. 1.

2.1 Efficient Massive Training Data Generation

One of the key reasons for the success of deep learning is its power to use huge amounts of training data, such as hundreds of thousands of data points. However, since CSPs are NP-hard, it is in general elusive to generate such a huge amount

of labeled data. In this subsection, we develop a new method that efficiently generates massive amounts of labeled data.

We generalize Model A [26] to create a random binary CSP generation model, henceforth referred to as generalized Model A. Model A generates a binary CSP as follows [26]. It independently selects each one of the $n(n-1)/2$ pairs of variables with a given probability p, and, for each selected pair of variables X_i and X_j, it marks each one of the $|D_i| \cdot |D_j|$ possible pairs of values as incompatible independently with a given probability q. Here, p characterizes how many constraints exist in a CSP, and q characterizes how restrictive the constraints are. In generalized Model A, q can vary from constraint to constraint (denoted by q_{ij} for the pair of variables X_i and X_j). Model A has an important property: It always generates CSPs that are unsatisfiable when $n \to \infty$ if $p, q > 0$ [2]. Generalized Model A also has this property if $p > 0$ and $\forall X_i, X_j \in \mathcal{X} : q_{ij} > 0$, since it generates CSPs that are more constrained than those generated by Model A with the same p and $q = \min_{X_i, X_j \in \mathcal{X}} q_{ij}$.

By making use of this property of generalized Model A, we are able to generate data points with a low mislabeling rate as follows. To generate a data point with label `UNSATISFIABLE`, we simply follow generalized Model A with non-zero p and q_{ij}'s. To generate a data point with label `SATISFIABLE`, we use the same procedure but update the compatibilities of tuples in generated constraints to allow for a randomly selected solution. We refer to this data generation method as *generalized Model A-based method* (GMAM). To avoid data imbalance, we generate comparable numbers of data points labeled `SATISFIABLE` and `UNSATISFIABLE`. Using this approach, we can efficiently generate huge amounts (such as millions) of labeled data points for training. The main intuitive reason that we use generalized Model A instead of Model A is that it leads to a distribution of CSPs that is more spread out and may be beneficial for training cNNs.

Although generalized Model A always generates CSPs that are unsatisfiable when $n \to \infty$ if $p > 0$ and $\forall X_i, X_j \in \mathcal{X} : q_{ij} > 0$, it is still desirable to have some bounds on the probability of mislabeling a CSP for finite n, which can be used to guide the choice of p and q_{ij}. Among all data points labeled `SATISFIABLE`, there are no mislabeled data points since a solution is guaranteed during data generation. For a data point labeled `UNSATISFIABLE`, we prove that:

Theorem 1. *Consider a data point with binary CSP $P = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$. If it is generated using GMAM and is labeled `UNSATISFIABLE`, the probability of it being mislabeled is no greater than $\prod_{X_i \in \mathcal{X}} |D(X_i)| \prod_{X_i, X_j \in \mathcal{X}} (1 - pq_{ij})$.*

Proof. The probability of mislabeling the CSP equals the probability that it has at least one solution, denoted by $P(n_{\text{sol}} \geq 1)$. Using Markov's inequality, we have $P(n_{\text{sol}} \geq 1) \leq \mathbb{E}(n_{\text{sol}})$, where $\mathbb{E}(n_{\text{sol}})$ is the expected number of solutions of the CSP. We also have

$$\mathbb{E}(n_{\text{sol}}) = \mathbb{E}\left(\sum_{a \in \mathcal{A}(\mathcal{X})} \mathbb{1}_{a \text{ is a solution}}\right) = \sum_{a \in \mathcal{A}(\mathcal{X})} \mathbb{E}\left(\mathbb{1}_{a \text{ is a solution}}\right) =$$
$$\sum_{a \in \mathcal{A}(\mathcal{X})} P(a \text{ is a solution}) = \sum_{a \in \mathcal{A}(\mathcal{X})} \prod_{X_i, X_j \in \mathcal{X}} (1 - pq_{ij}) = \prod_{X_i \in \mathcal{X}} |D(X_i)| \prod_{X_i, X_j \in \mathcal{X}} (1 - pq_{ij}),$$

where $A(\mathcal{X})$ is the set of all assignments of values to variables in $\mathcal{X}$. Therefore, the probability that a data point labeled `UNSATISFIABLE` is mislabeled is no greater than $\prod_{X_i \in \mathcal{X}} |D(X_i)| \prod_{X_i, X_j \in \mathcal{X}} (1 - pq_{ij})$. □

2.2 Training and Prediction on General CSP Datasets

Applying a deep NN to a small dataset directly may cause overfitting due to the large number of training parameters. Although we can use GMAM to efficiently generate huge amounts of training data, training a deep NN on a dataset from a distribution different from the dataset of interest usually does not lead to good results, even if the training dataset is huge. To overcome this issue, there are two common classes of techniques: *domain adaptation* and *data augmentation*.

Domain adaptation refers to learning from one source of data and predicting on a different source of data with a different distribution, due to the scarcity of available labeled data from the latter source. By using domain adaptation techniques, a small set of labeled data of interest, assumingly generated from an arbitrary distribution different from generalized Model A, can still be made viable. In particular, we can train on a mix of these available data and data generated using GMAM, and then evaluate on the test data of interest.

Data augmentation refers to transforming data without changing their labels, known as *label-preserving transformations*. For example, in object recognition tasks in computer vision applications, to generate more training data, we can augment an image by translating or reflecting horizontally without changing its label [17]. In the context of CSP-cNN, we can augment an input CSP by changing the order of variables or their domain values, i.e., exchanging their corresponding rows and columns of the CSP matrix. This does not alter the satisfiabilities of the CSPs and therefore does not change their labels.

3 Experimental Evaluation

We evaluated CSP-cNN and the relevant methods mentioned above experimentally. We used Keras [7] with the TensorFlow [1] backend, that uses the GPU to accelerate forward and backward propagation to implement NNs.

Evaluation on Data Generated Using GMAM Using GMAM, we generated 200,000 training data points, 10,000 validation data points, and 10,000 test data points. Each data point is a binary CSP of 128 Boolean variables limited by the computational capacity of our hardware and the size of our CSP-cNN. For each data point, we randomly chose p and all q_{ij}'s between 0.12 and 0.99. Theorem 1 guarantees that the probability of mislabeling an `UNSATISFIABLE` data point is $\leq 2^{128} \times (1 - 0.12 \times 0.12)^{128 \times (128-1)/2} = 2.14 \times 10^{-13}$. Half of the data points in each of the training, validation, and test datasets are labeled `SATISFIABLE` and the others `UNSATISFIABLE`.

We first trained our CSP-cNN using the training data generated above. We initialized all parameters using *He-initialization* [14]. We trained our CSP-cNN

Table 1: Test accuracies on GMAM generated data.

	CSP-cNN	NN-image	NN-1	NN-2	M
Acc (%)	>99.99	50.01	98.11	98.66	64.79

Table 2: First two columns show test accuracies of CSP-cNN in all 3 rounds of cross validation. Last column shows the test accuracy of CSP-cNN (trained on GMAM generated data) on MMEM generated data.

Data Trained	Mixed	MMEM	GMAM
Acc (%)	100.00/100.00/100.00	50.00/50.00/50.00	50.00

using *stochastic gradient descent* (SGD) with a mini-batch size of 128 for 59 epochs. In each epoch, we randomly shuffled all data points. We used a *learning rate* of 0.01 for the first 5 epochs and a learning rate of 0.001 for the last 54 epochs. We used *binary cross entropy* as the loss function. Each epoch took about 520 seconds to finish on a GPGPU "NVIDIA(R) Tesla(R) K80."

After training, all training, validation, and test accuracies were greater than 99.99%. Therefore, we conclude that, while constraint tightnesses and constraint densities do not determine the satisfiabilities of CSPs, deep NNs, such as our CSP-cNN, can be capable of accurate predictions when a huge amount of training data are available, at least on Boolean binary CSPs.

To further demonstrate the effectiveness of our CSP-cNN on GMAM generated data, we also compared our CSP-cNN with three other NNs. The first NN, referred to as NN-image, had the same architecture as our CSP-cNN, but its input was a gray-scale image converted from the ASCII codes of its input text file as described in [19]. The other two NNs were *plain*, i.e., had only fully connected hidden layers. The first plain NN, referred to as NN-1, had only 1 fully connected hidden layer with 256 ReLUs and 1 output layer with a single neuron with a sigmoid activation function, i.e., the last two layers of our CSP-cNN. This is a classical shallow NN architecture. The second plain NN, referred to as NN-2, was constructed by inserting 1 more fully connected hidden layer with 1024 ReLUs after the input layer in NN-1. Both plain NNs used the same parameter initialization and regularization as our CSP-cNN. We trained NN-image using a training procedure similar to that of CSP-cNN except that it used one more epoch with a learning rate of 0.01. We trained both plain NNs for 120 epochs using SGD with a learning rate of 0.01 for the first 60 epochs and 0.001 for the last 60 epochs. They both used a mini-batch size of 128.

Our experimental results are shown in Table 1. The test accuracy of our CSP-cNN was better than those of NN-1 and NN-2 and far better than that of NN-image. Thus, the CSP matrix of a CSP seems to provide a better input representation than the approach in [19] and seems to reveal useful structure of the CSP. We also compared with an approach that predicts a CSP's satisfiability using its number of incompatible tuples, referred to as "M" in Table 1. It selects a threshold and predicts CSPs with a number of incompatible tuples above this threshold to be unsatisfiable and other CSPs to be satisfiable. The best threshold for the test data is 13435 and led to an accuracy of 64.79%.

Evaluation of Domain Adaptation and Data Augmentation Due to the lack of small ($n \leq 128$) benchmark instances of Boolean binary CSPs where satisfiabilities need to be determined, we randomly generated 1,200 binary CSPs with

128 Boolean variables using a model similar to Model E [2]. We generated only 1,200 CSPs to mimic most real-world scenarios where labeled CSPs are costly to obtain. We generated these CSPs as follows. We divided all CSPs into two groups. For each CSP in the first group, (a) we divided the 128 Boolean variables into two partitions, with 64 variables each; (b) for each pair of variables from different partitions, we randomly added a binary constraint between them with probability 0.99; (c) within each constraint, we randomly marked exactly 2 (out of 4) tuples as incompatible. For each CSP in the second group, we generated it using a similar approach, except that, in Step (c), we also guaranteed that incompatible tuples do not rule out a randomly generated solution (while it remains that exactly 2 tuples are incompatible in each constraint). We refer to this random CSP generation method as *Modified Model E-based method* (MMEM). Using Choco [22], we labeled 600 CSPs `SATISFIABLE` and 600 `UNSATISFIABLE`.

The distribution of CSPs resulting from using MMEM is different from that of the ones resulting from using GMAM, for the following reasons. There are formally proven significant differences between the asymptotic satisfiability properties of Model A and Model E [2]. Step (a) yields a bipartite variable interaction structure. Step (c) guarantees the same tightness in each constraint, which makes the satisfiabilities of the CSPs unrecognizable from their tightnesses. For these reasons, these random CSPs suffice for a proof of concept.

We evaluated the effectiveness of domain adaptation and data augmentation for our CSP-cNN on the CSPs generated by MMEM using stratified 3-fold cross validation, i.e., we divided the 1,200 CSPs into 3 sets with equal numbers of satisfiable and unsatisfiable CSPs. Since there are only 400 data points in each of these 3 sets, for each data point in the training set, we used the augmentation method in Section 2.2 124 times to produce 124 more data points each. Therefore, in each round of cross validation, we used $125 \times 800 = 100,000$ training data points. We mixed these 100,000 data points with the 200,000 data points generated using GMAM and trained our CSP-cNN on them. We used SGD and trained for 30 epochs. We used a learning rate of 0.01 in the first 10 epochs and a learning rate of 0.001 in the last 20 epochs. As a baseline, we also trained CSP-cNN by augmenting each data point for 324 times so that the number of training data points was also $325 \times 800 = 300,000$ in each round. The training procedure was the same. We also directly applied the CSP-cNN previously trained on GMAM generated data to all 1,200 data points.

Our experimental results are shown in Table 2. Our mixed data points produced test accuracies of 100% in all three rounds of cross validation. On the other hand, CSP-cNNs trained only on augmented MMEM or GMAM generated data have high test errors in (cross) validation and always produced the same prediction regardless of their input. When training our CSP-cNN only on augmented MMEM generated data, we were unable to reduce the test error even by tuning hyperparameters, such as the learning rate, initialization, and the optimization algorithm. This shows that our GMAM generated data seem to play a key role in enabling effective deep learning for CSPs via domain adaptation. To further confirm this, we ran similar experiments with various percentages

Table 3: Test accuracies of all three rounds of cross validation for different percentages of MMEM generated data when domain adaptation is used.

Percentage of MMEM (%)	0.00	33.33	36.00	40.00	46.66	53.33	66.67	70.67	78.67	100.00
Average Accuracy (%)	50.00	100.00	100.00	83.33	66.67	83.33	66.67	66.67	50.00	50.00

of MMEM generated data in the training data by varying the number of times each data point is augmented. In these experiments, GMAM generated data points were randomly selected to fill the total number of training data points to 300,000. Table 3 shows our experimental results. When MMEM generated data points constituted 33.33–36.00% of the training data, the average test accuracies reached 100%. However, when MMEM generated data points constituted more than 40.00% of the training data, the test accuracies became lower and unstable.

4 Conclusions and Future Work

In this paper, we effectively applied our CSP-cNN, a deep NN architecture, to predict satisfiabilities of CSPs with prediction accuracies higher than 99.99%. To the best of our knowledge, this is the first effective application of deep learning to random CSPs whose constraint tightnesses and constraint densities do not determine their satisfiabilities. Due to the NP-hardness of CSPs, training data are usually too scarce to be effectively used by deep learning. We addressed this issue by generating huge amounts of labeled data using GMAM. We experimentally demonstrated the high effectiveness (>99.99% test accuracy) of applying our CSP-cNN to these data on random Boolean binary CSPs. While these CSPs are known to be in P, we used them as an initial demonstration. For CSPs drawn from a distribution different from that of GMAM, we once again addressed the issue of lack of training data. We did this by augmenting the training data and mixing them with GMAM generated CSPs. Finally, we experimentally demonstrated the superior effectiveness of these techniques on MMEM generated CSPs.

So far, we have only experimented on small easy random CSPs that were generated in two very specific ways. One future research direction is to understand the generality of our approach, for example, by experimenting on larger, hard, and real-world CSPs, analyzing what our CSP-cNN learns, and evaluating how robust our approach is with respect to the training data and hyperparameters. A second future research direction is to understand exactly how our approach should be used, for example, how the effectiveness of our CSP-cNN depends on the amount of available training data and the amount of data augmentation used to increase them. A third future research direction is to generalize our CSP-cNN to accommodate more types of constraints. (a) For non-binary table constraints, we could naively increase the dimensionality of the CSP matrix to be equal to the maximum arity of the constraints. A more practical method might be to represent input CSPs as constraint graphs and adapt the graph representation methods in [20]. (b) For symmetric global constraints, we could adapt the methods that apply recurrent NNs (rNNs) to Boolean satisfiability [25]. Then, an NN architecture that combines cNNs and rNNs could be used.

The research at the University of Southern California (USC) was supported by National Science Foundation (NSF) under grant numbers 1724392, 1409987, and 1319966.

References

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: Large-scale machine learning on heterogeneous systems (2015), https://www.tensorflow.org/, software available from tensorflow.org
2. Achlioptas, D., Molloy, M.S.O., Kirousis, L.M., Stamatiou, Y.C., Kranakis, E., Krizanc, D.: Random constraint satisfaction: A more accurate picture. Constraints 6(4), 329–344 (2001)
3. Amadini, R., Gabbrielli, M., Mauro, J.: An empirical evaluation of portfolios approaches for solving CSPs. In: the International Conference on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming. pp. 316–324 (2013)
4. Amadini, R., Gabbrielli, M., Mauro, J.: An enhanced features extractor for a portfolio of constraint solvers. In: the Annual ACM Symposium on Applied Computing. pp. 1357–1359 (2014)
5. Arbelaez, A., Hamadi, Y., Sebag, M.: Continuous search in constraint programming. In: the IEEE International Conference on Tools with Artificial Intelligence. pp. 53–60 (2010)
6. Bourlard, H.A., Morgan, N.: Connectionist Speech Recognition. Springer (1994)
7. Chollet, F., et al.: Keras. https://keras.io (2015)
8. Ciregan, D., Meier, U., Schmidhuber, J.: Multi-column deep neural networks for image classification. In: the IEEE Conference on Computer Vision and Pattern Recognition. pp. 3642–3649 (2012)
9. Galassi, A., Lombardi, M., Mello, P., Milano, M.: Model agnostic solution of CSPs via deep learning: A preliminary study. In: the International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research. pp. 254–262 (2018)
10. Gent, I.P., Jefferson, C., Kotthoff, L., Miguel, I., Moore, N.C., Nightingale, P., Petrie, K.: Learning when to use lazy learning in constraint solving. In: the European Conference on Artificial Intelligence. pp. 873–878 (2010)
11. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016)
12. Guerri, A., Milano, M.: Learning techniques for automatic algorithm portfolio selection. In: the European Conference on Artificial Intelligence. pp. 475–479 (2004)
13. Hahnloser, R.H.R., Sarpeshkar, R., Mahowald, M.A., Douglas, R.J., Seung, H.S.: Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. Nature 405, 947–951 (2000)
14. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In: the IEEE International Conference on Computer Vision. pp. 1026–1034 (2015)
15. Kadioglu, S., Malitsky, Y., Sellmann, M., Tierney, K.: ISAC – instance-specific algorithm configuration. In: the European Conference on Artificial Intelligence. pp. 751–756 (2010)
16. Kotthoff, L.: Algorithm selection for combinatorial search problems: A survey. In: Data Mining and Constraint Programming: Foundations of a Cross-Disciplinary Approach, pp. 149–190 (2016)

17. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: the Neural Information Processing Systems Conference. pp. 1097–1105 (2012)
18. LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., Jackel, L.D.: Backpropagation applied to handwritten zip code recognition. Neural Computation 1(4), 541–551 (1989)
19. Loreggia, A., Malitsky, Y., Samulowitz, H., Saraswat, V.: Deep learning for algorithm portfolios. In: the AAAI Conference on Artificial Intelligence. pp. 1280–1286 (2016)
20. Niepert, M., Ahmed, M., Kutzkov, K.: Learning convolutional neural networks for graphs. In: the International Conference on Machine Learning. pp. 2014–2023 (2016)
21. O'Mahony, E., Hebrard, E., Holland, A., Nugent, C., O'Sullivan, B.: Using case-based reasoning in an algorithm portfolio for constraint solving. In: the Irish Conference on Artificial Intelligence and Cognitive Science (2008)
22. Prud'homme, C., Fages, J.G., Lorca, X.: Choco Documentation. TASC - LS2N CNRS UMR 6241, COSLING S.A.S. (2017), http://www.choco-solver.org
23. Pulina, L., Tacchella, A.: A multi-engine solver for quantified Boolean formulas. In: the International Conference on Principles and Practice of Constraint Programming. pp. 574–589 (2007)
24. Sateesh Babu, G., Zhao, P., Li, X.L.: Deep convolutional neural network based regression approach for estimation of remaining useful life. In: the International Conference on Database Systems for Advanced Applications. pp. 214–228 (2016)
25. Selsam, D., Lamm, M., Bünz, B., Liang, P., de Moura, L., Dill, D.L.: Learning a SAT solver from single-bit supervision. arXiv:1802.03685 [cs.AI] (2018)
26. Smith, B.M., Dyer, M.E.: Locating the phase transition in binary constraint satisfaction problems. Artificial Intelligence 81(1), 155–181 (1996)
27. Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K.: SATzilla: Portfolio-based algorithm selection for SAT. Journal of Artificial Intelligence Research 32, 565–606 (2008)