Automated SAT Problem Feature Extraction using Convolutional Autoencoders

Marco Dalla
SFI Centre for Research Training in AI, University College Cork, Ireland

Andrea Visentin
School of Computer Science & IT, University College Cork, Ireland

Barry O'Sullivan
Insight Centre for Data Science and Analytics, University College Cork, Ireland

This paper was previously published in the proceedings of ICTAI 2021 [8]

Abstract

The Boolean Satisfiability Problem (SAT) was the first known NP-complete problem and has a very broad literature focusing on it. It has been applied successfully to various real-world problems, such as scheduling, planning and cryptography. SAT problem feature extraction plays an essential role in this field. SAT solvers are complex, fine-tuned systems that exploit problem structure. The ability to represent/encode a large SAT problem using a compact set of features has broad practical use in instance classification, algorithm portfolios, and solver configuration. The performance of these techniques relies on the ability of feature extraction to convey helpful information. Researchers often craft these features "by hand" to capture particular structures of the problem. Instead, in this paper, we extract features using semi-supervised deep learning. We train a convolutional autoencoder (AE) to compress the SAT problem into a limited latent space and reconstruct it minimizing the reconstruction error. The latent space projection should preserve much of the structural features of the problem. We compare our approach to a set of features commonly used for algorithm selection. Firstly, we train classifiers on the projection to predict if the problems are satisfiable or not. If the compression conveys valuable information, a classifier should be able to take correct decisions. In the second experiment, we check if the classifiers can identify the original problem that was encoded as SAT. The empirical analysis shows that the autoencoder is able to represent problem features in a limited latent space efficiently, as well as convey more information than current feature extraction methods.

2012 ACM Subject Classification • SAT solving • Feature selection

Keywords and phrases boolean satisfiability; deep learning; convolutional autoencoders; feature extraction; satisfiability prediction; CNF encoding

Digital Object Identifier 10.4230/LIPIcs...

Related Version http://arxiv.org/abs/2106.07162

Funding This publication has emanated from research conducted with financial support of Science Foundation Ireland under Grant 16/RC/3918, 12/RC/2289-P2, and 18/CRT/6223, which are co-funded under the European Regional Development Fund.

1 Introduction

A SAT instance consists of a Boolean formula that involves a set of Boolean variables connected by the logical operators "and", "or" and "not". The Propositional Boolean Satisfiability problem involves finding an assignment to the variables such that the formula evaluates to true (satisfiable instance) or proves that such an assignment does not exist (unsatisfiable instance). SAT solvers are software designed to take such instances and find a solution or prove that one does not exist. Many studies have been dedicated to developing new solving techniques and improving existing ones. These solvers compete annually in the SAT

competition [4] to test the state of the art in this discipline on hard problem instances. These solvers use different solving techniques, such as Conflict-Driven Clause Learning (CDCL), Look-Ahead SAT, and Stochastic Local Search. In these competitions it is clear that no solver outperforms all others on all problems; depending on the instance type, their rank varies consistently. For example, CDCL solvers are especially efficient in solving industrial SAT instances. The reason behind this specialization is not yet clear. Due to this, portfolio algorithms often perform best. A portfolio in this setting is a collection of individual solvers that can be selectively deployed depending on the type of instance that has to be tackled. Portfolios deploy a multi-class classifier on the instance features to select the best solver. This task is called the Algorithm Selection Problem [16].

Another approach to tackle a SAT instance is to predict the satisfiability as a classification problem using machine learning. The first attempt in this field [9] uses features extracted by a portfolio algorithm. Recently, the pervasiveness of deep learning extended to this field as well. The most closely related attempt is NeuroSAT [19], a graph deep learning network that iteratively tries to classify a graphical representation of the SAT instance. This approach can be seen as a deep learning heuristic solver.

A fundamental component of these approaches is the extraction of meaningful features from a SAT instance, e.g. [2]. This task aims to represent a problem that involves a high number of variables and clauses into a handful of values preserving the relevant information on its structure and properties. The features are generally statistical information of the instance, e.g. the number of variables, clauses or their ratio, or computed on different representations of the problem, such as the graph encoding. The classic process of extracting features mirrors the human view of the problem structure. One of the reasons why it is hard to understand the behaviour of the solvers on different types of problems is that we are trying to see them from a human perspective. Machine-automated feature extraction can reduce the human bias on the problem description.

In this work, we present a semi-supervised deep learning automated approach to extract features from SAT instances. We train a convolutional autoencoder (AE) to compress and reconstruct an instance minimizing the reconstruction error. An AE is a deep network that takes the input and learns to copy it, minimizing the differences between the original and the network's output. The particularity is that one of the internal layers has a limited number of neurons, acting as an information bottleneck. The AE learns to compress the relevant information to fit through this bottleneck and reconstruct them correctly. It can be decomposed into an encoder and a decoder; the encoder projects the input into the latent space represented by the bottleneck, while the decoder takes a point of the latent space and tries to reconstruct the original instance. During the training phase, the AE learns a low dimensional representation of the input; if this representation preserves the information, a classifier should be able to take decisions on these features.

2 Encoding

Conjunctive normal form (CNF) is the instance format used by the vast majority of the solvers. A CNF is composed of a conjunction of *clauses*. Each clause is a disjunction of *literals*. A literal is the occurrence of a variable negated or not. All SAT problems can be transformed into CNF formulas in linear time with only a linear increase in the formula size, with preserving only satisfiability [21]. The DIMACS CNF format is the standard, however, it is not well suited for an AE. Different approaches have been introduced, such as transforming it into an image [15] or using a graph representation[6]. The network has

to output a multi-class classification. One-hot encoding is the general approach for such problems.

Our approach transforms a CNF into an equivalent binary array. Given a CNF with C clauses and V variables, we encode each clause i as an array of binary variables $\mathbf{x}^i$. For each variable j, $x^i_{2j} = 1$ if j appears in clause i, and $x^i_{2j+1} = 1$ if j appears negated in that clause, otherwise they take value 0. Thereby each CNF can be encoded in $2CV$ binary variables. We fixed a maximum number of clauses and variables encoding all instances according to those sizes to make input size constant.

2.1 Symmetry breaking

A drawback of CNF formulas is that they are subjected to different invariances. These symmetries make the network learning process harder since the network needs to learn that different inputs might encode an equivalent state of the problem. Graph representation removes some of these symmetries [18]. Invariances are particularly detrimental in our case due to the loss computation. A reconstructed instance that represents the same problem but with variables and clauses that are permuted has a really high reconstruction loss. In our case, we considered an ordering that reduces the invariances of our encoding.

Many operations can change the CNF without affecting its satisfiability. We tackled them in the following way:

- *Variable negation*, for example, negating all the occurrences of a given variable. For each variable, the normal literal has to appear at least as much as the negated one.
- *Permuting variables*, for example, swapping all the occurrences of variable $j = 1$ with variable $j = 2$. We ordered them by the number of the literal appearances. The first variable is the one that appears the most in the formula. Ties are broken considering the variable with the minimum amount of negated literals.
- *Permuting literals in a clause*, changing the order of the literals in a clause does not affect the solution. Our encoding is not affected by this symmetry.
- *Permuting clauses*, for example, swapping the first clause with the second one. We order the clauses by their cardinality (number of literals appearing in the clause). Ties are broken considering the literal with the lower index.

While this approach does not resolve some invariances completely, it allows a network to learn patterns. For example, the variable that appears the most appears negated in a clause with a variable that rarely appears.

2.2 Dataset

The main reason for generating the dataset is the necessity for a sizeable number of instances with diverse structures that are well balanced between SAT and UNSAT. Deep learning solution quality strongly depends on the quantity and how representative the training dataset. Moreover, the use of deep learning algorithms and the binary encoding presented in the previous section makes handling large instances quite complicated in terms of memory usage; therefore, a reduced number of variables and clauses was required. To the best of our knowledge, no publicly available dataset is satisfying these constraints, so we opted for generating one. Creating a generator for problem instances that have the structure and computational properties that are more similar to real-world instances has been an ongoing challenge [11]. The dataset used in the empirical section of this paper is generated using CNFgen [14]. This tool produces propositional formulas in the CNF DIMACS format that can be used as a benchmark for SAT solvers. It features several formula families (e.g. pigeonhole

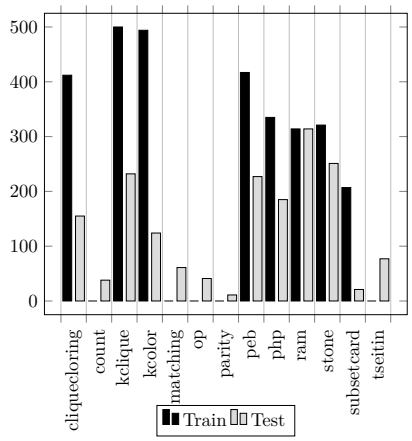


Figure 1 Distribution of the different SAT instances for the training dataset and testing dataset

principle, ordering principle, k-coloring, etc.) as well as several formula transformations and the possibility of producing formulas directly from graph structures.

Two datasets were assembled. The first one, which we will later refer to as the training dataset, consists of 3000 instances uniformly distributed in eight types, mainly stemming from graph structures. Of these 3000 instances, 1562 are proven unsatisfiable and 1438 satisfiable beforehand using the Glucose SAT solver [3]. This set is used to train the AE in order to find the optimal representation in the latent space. The learned embedding is then used to train a machine learning classifier to predict both the satisfiability and the instance type.

A second dataset, for testing, is generated to assess how well the autoencoder performs on previously unseen instances. This set consists of 1737 instances divided into 12 classes. Out of those 12 classes, 8 are of the same type used for the training dataset, while the remaining 5 are new types of instances. As previously, we determine the satisfiability of the instances using the Glucose solver, resulting in 1035 unsatisfiable samples and 702 satisfiable samples. In the empirical experiments, the training set is used as a whole, while the test set is considered both as a whole and divided between previously seen and new instance types. This separation aims to analyse the ability of the AE to generalise to unseen data structures. The distribution of both datasets is shown in Figure 1.

3 Method

The deep learning algorithm chosen for the feature extraction is a convolutional autoencoder (AE) [13]. It is a type of feed-forward neural network architecture that leverages the technique of representation learning. Specifically, the network is composed of two branches that function as an encoder-decoder pair. At the conjunction, a bottleneck is present to obtain a compressed knowledge representation of the input. The bottleneck compression can be referred to as a latent space of lower dimensionality into which the input is projected. This technique is based on the assumption that the input data has a structure that can be efficiently learned and used in its reduced form to reproduce the input. Indeed, the task of an AE is to generate an output that is a reconstruction of the original input. The **reconstruction loss** $\mathcal{L}(\tilde{x}, x)$ measures how well the original input has been reproduced in the output from the latent space. In this case, the chosen reconstruction loss was the *binary-crossentropy* [17]; this loss considers every binary variable of the encoding as a binary classification task. This

choice is possible due to the binary encoding of the input presented in the previous section. Before feeding the input into the network, it undergoes the encoding and symmetry breaking process.

Using convolutional layers offers many advantages over a dense AE. These layers are widely used in image and signal processing. They can recognize patterns even when their position is moved around the input. We can use them to learn literals and clause patterns. Convolutional layers can process bigger inputs compared to fully connected layers. Finally, it can focus the scope of a literal to the clause it is part of. In previous works using convolutional neural networks, such as [15], a window includes literals from consecutive clauses, learning patterns on those.

The encoder part of the network takes the input and projects it into the latent space. It starts with a convolutional layer with a single filter that encodes a literal, with windows of size 2 that do not overlap. This layer converts each literal to a single value. We prefer this approach over using a three-valued encoding for each variable in each clause (such as 0 for non-present, 1 for present, and 2 for present and negated) because this encoding would imply different errors in the case of the case misclassification. The second convolutional layer has non-overlapping windows with a size equal to the maximum number of variables. Each window covers all the literals involved in a clause without overlapping to literals in other clauses. The advantage is that it can learn patterns at clause level, e.g. clauses that involve the most used variables all negated. Finally, we deploy a set of convolutional layers. Using a series of convolutional layers helps to learn higher-level patterns, and it generally outperforms both in training times and performances a single layer with more filters. The encoder ends with a short series of fully connected layers, the last one with a number of neurons equal to the latent space size.

These layers, with the exception of the final one, are followed by dropout [20] and batch-normalization [10] layers. These make the algorithm less sensitive to overfitting, reduce the internal covariant shift and allow a higher learning rate without the possibility of gradient explosion or vanish.

The decoder mirrors the encoder structure with transposed convolutions. We select a latent space of size $\delta = \{8, 16, 24, 32, 40\}$. The optimizer Adam [12] is used to speed up the back-propagation. The initial learning rate is set to 10^{-4} with a decay of 10^{-5}. Finally, the Rectified Linear Unit [1] is chosen as the activation function. Figure 2 shows the entire structure of the AE implemented and used in the experiments.

Before the choosing of the optimal parameters and the AE structure, a large hyperparametrisation was conducted. We trained all the networks for multiple days on three NVIDIA Quadro RTX 8000 graphic cards.

4 Experimental results

The empirical evaluation aims to compare the features extracted using the approach presented in this paper to the widely used SatZilla features[22]. SATZilla is a portfolio SAT solver that for each instance selects the solver based on a set of 127 features that includes different aspects of the problem: size, graph, balance and timing features. Of this set of features we ignored the timing/probing ones that are solver dependent and not specific of the CNF. The goal of a compression/feature extraction solution is to preserve/extract relevant information. To assess the importance of the information extracted, a common approach is to train a classifier on the features; if a classifier achieves a high accuracy it means that the features extract enough information to take a correct decision. Firstly, we check if the features can be

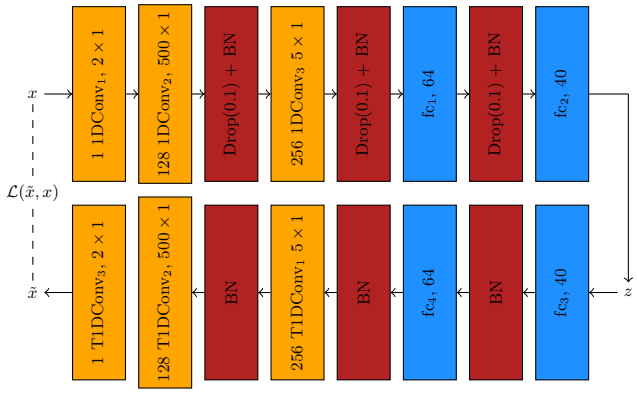


Figure 2 Convolutional Autoencoder consisting of encoder (right) and decoder (left). The encoder features 3 convolutional layers, followed by 2 fully connected layers. It also features dropout and batch normalization layers. The decoder mirrors the encoder structure, with transpose convolutions instead. For the convolutional layers, the number of filters and window sizes are specified, for the dense layer the number of neurons.

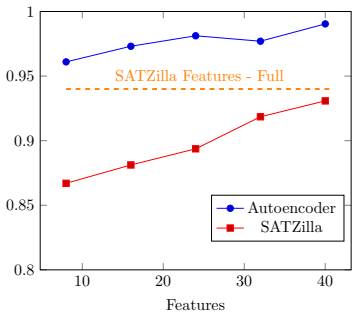
used for the binary classification of the satisfiability of the problem. In a second experiment, we build a classifier that is able to recognize the type of graph problem encoded. We use the dataset presented in Section 2.2. We train the AE using an 80/20 validation split on the train dataset; we then use it to extract the features of both train and test set. Since SATZilla does not have a training phase its deployment is equivalent on the train and test set. We presents the results of XGBoost[7] as classifier. Then, the average accuracy is computed over 10 shuffles of 10 cross validation. We also analyse the performance on the training set. The AE is optimised for the instance compression, not for the classification task; comparing its performances with the test set we can understand how well the features generalise to unseen structures.

4.1 SAT/UNSAT classification

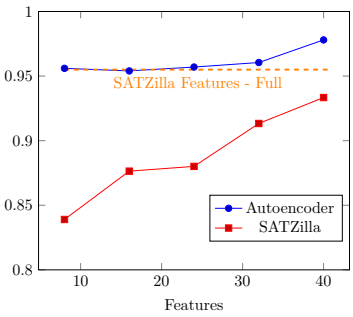
In this experiment, we try to predict the satisfiability of a problem based on the extracted features. This task is widely studied in the literature We project all the instances of the dataset into increasing size latent spaces, $\delta = \{8, 16, 24, 32, 40\}$. The goal is to test the ability of the approach to compress the instance structure. We train AEs for the different δ values on the training set. For the SATZilla features, we compress them using principal component analysis (PCA) to reduce their cardinality.

Figure 3a and Figure 3b show the results of our experiment on the train and the test set, respectively. The orange dotted line is the accuracy achieved by using the full SATZilla features set. The AE outperforms the best SATZilla in the training set even when compressing the SAT instance to just eight features, while in the test set that also contains different types of problems is comparable. When projecting to 40 dimensions the gap between the two increases.

It is clear that the PCA compression of the SATZilla features reduces the ability of the classifier to recognize the UNSAT instances. For the AE, when increasing the dimension of the latent space the accuracy improves similarly in both classes. We have similar results in the training set, where the best AE classify all the SAT instances correctly and makes mistakes in just a few UNSAT ones. To evaluate if our approach can generalise to new types



(a) SAT/UNSAT classification accuracy (training set).



(b) SAT/UNSAT classification accuracy (test set).

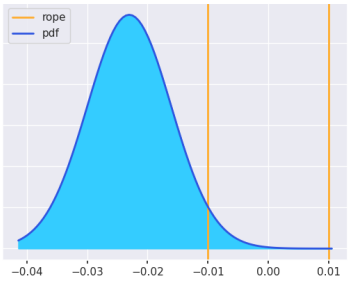


Figure 4 Posterior distribution of the comparison between AE 40 features and SATZilla full feature set. *rope* is the region of practical equivalence.

of instances we divided the test set in instance types that are present in the training set as well, and the ones that are not.

To understand if these differences are statistically significant, we study the performances of the best solutions of the two approaches using the Bayesian classifier comparison presented in [5]. This analysis allows us to compute the posterior probability of a classifier being better than the other and the probability of them being equivalent from a practical point of view (1% accuracy of difference). Figure 4 shows the results. The AE approach has the 96.73% probability to be better than SATZilla in SAT/UNSAT classification on the test set, and the 3.27% of being practically equivalent. For the sake of brevity, we omitted the plot computed on the training set where the AE has a 99.99% probability of being practically better than the competitor.

The algorithm presented in this work computes features that preserve the instance satisfiability information in a better way compared to the existing approaches. Increasing the compression level marginally affect the performances of the AE, but it has a consistent impact on the SATZilla features. The ability to extend these results to unseen instances with different structure (originated from different problems) shows the generalization abilities of SATZilla features.

4.2 Instance classification

One of the most efficient approaches to solve SAT instances is using portfolio solvers. These solvers use a collection of normal SAT solvers and select a specific one depending on the instance. SATZilla features are designed for this purpose, while AE features aim to represent the whole instance in a limited projected space. In this experiment, we tried to identify the original problem class encoded as a CNF instance using the classes presented in [14]. We

Table 1 instance classification accuracy

Instance Type	Training set		Test set - Old		Test set - New		Test set - Full	
	Autoencoder	SATZilla	Autoencoder	SATZilla	Autoencoder	SATZilla	Autoencoder	SATZilla
cliquecoloring	73	95	58	76	-	-	58	91
count	-	-	-	-	50	67	50	64
kclique	86	97	73	80	-	-	67	89
kcolor	95	100	67	86	-	-	79	93
matching	-	-	-	-	86	75	96	87
op	-	-	-	-	100	88	100	80
parity	-	-	-	-	98	91	98	87
peb	93	100	90	100	-	-	96	100
php	80	95	97	100	-	-	89	100
ram	100	100	100	100	-	-	100	100
stone	97	97	100	97	-	-	88	89
subsetcard	50	38	65	63	-	-	93	85
tseitin	-	-	-	-	75	100	75	97
Average	**85.9**	**93.4**	**86.1**	**94.3**	**82.6**	**91.8**	**84.5**	**93.3**

used the same features extracted in the previous experiment.

Table 1 shows the classification accuracy divided by class for the AE with 40 latent dimensions and the complete set of SATZilla features. The classes not present on the set are left empty. We analyse the test set complete, and divided between old and new instance types. SATZilla outperforms the AE in this task. We assume this is because some instances have similar structures but differ for some statistical features. For example, the AE struggles to distinguish the graph-based problems (cliquecoloring, kclique, kcolor). Likely, these problems have similar structures, and their differences are not well represented in the projected space. For some instances, the AE has better performances; for the training set in subset-cardinality, and for the test set in matching, op, parity. SATZilla features are better for identifying the type of instance. We believe that these problems are better represented by the compression approach we use.

5 Conclusions

In this paper, we presented a new automated approach to extract features from a SAT instance. In our approach, the features are learned by an unsupervised neural network and not crafted by a human. To do so, we used a convolutional autoencoder.

The empirical study shows that our approach can preserve the instance information to allow a classifier to predict the satisfiability of the problem and the type of instance. Compared to the state of the art, our approach can convey more information in a limited feature space. The difference between the two approaches is statistically significant and relevant from a practitioner point of view. SATZilla features outperform the presented approach on instance type classification. However, the results show that the information conveyed by the AE can help to identify types of instances in which the statistical features struggles.

Our work opens a variety of research questions. A deep analysis of the latent space can correlate its dimensions to the artificial features, this might allow us to discover blind spots in the statistical features currently in use. We plan to improve the results further by introducing reconstruction error losses tailor-made for SAT instances. Another aspect we want to cover is creating a set that comprises both human-designed and deep learning computed features to encode different aspects of the SAT instances. Finally, we intend to investigate the possibility of using deep learning generative models such as variational autoencoders.

References

1 Abien Fred Agarap. Deep learning using rectified linear units (relu). *ArXiv*, abs/1803.08375, 2018.

2 Carlos Ansótegui, Maria Luisa Bonet, Jesús Giráldez-Cru, and Jordi Levy. Structure features for sat instances classification. *Journal of Applied Logic*, 23:27–39, 2017. doi:10.1016/j.jal.2016.11.004.

3 Gilles Audemard and Laurent Simon. On the glucose sat solver. *Int. J. Artif. Intell. Tools*, 27:1840001:1–1840001:25, 2018.

4 Tomáš Balyo, Nils Froleyks, Marijn JH Heule, Markus Iser, Matti Järvisalo, and Martin Suda. Proceedings of sat competition 2020: Solver and benchmark descriptions. 2020.

5 Alessio Benavoli, Giorgio Corani, Janez Demsar, and Marco Zaffalon. Time for a change: a tutorial for comparing multiple classifiers through bayesian analysis. *J. Mach. Learn. Res.*, 18:77:1–77:36, 2017.

6 Benedikt Bünz and Matthew Lamm. Graph neural networks and boolean satisfiability. *arXiv*, pages 1–9, 2017.

7 Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

8 Marco Dalla, Andrea Visentin, and Barry O'Sullivan. Automated sat problem feature extraction using convolutional autoencoders. In *2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 232–239, 2021. doi:10.1109/ICTAI52525.2021.00039.

9 David Devlin and Barry O'Sullivan. Satisfiability as a classification problem. *Proc. of the 19th Irish Conf. on Artificial Intelligence and Cognitive Science*, 2008.

10 Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of ICML*, pages 448–456, 2015.

11 Henry Kautz and Bart Selman. Ten challenges redux: Recent progress in propositional reasoning and search. In *Proceedings of CP*, pages 1–18, 2003.

12 Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of ICLR*, 2015. URL: http://arxiv.org/abs/1412.6980.

13 Mark A. Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal*, 37(2):233–243, 1991. doi:10.1002/aic.690370209.

14 Massimo Lauria, Jan Elffers, Jakob Nordström, and Marc Vinyals. Cnfgen: A generator of crafted benchmarks. In *Proceedings of SAT*, pages 464–473, 2017.

15 Andrea Loreggia, Yuri Malitsky, Horst Samulowitz, and Vijay A. Saraswat. Deep learning for algorithm portfolios. In *Proceedings of AAAI*, pages 1280–1286, 2016.

16 John R. Rice. The algorithm selection problem. *Adv. Comput.*, 15:65–118, 1976. doi:10.1016/S0065-2458(08)60520-3.

17 Usha Ruby and Vamsidhar Yendapalli. Binary cross entropy with deep learning technique for image classification. *International Journal of Advanced Trends in Computer Science and Engineering*, 9, 10 2020. doi:10.30534/ijatcse/2020/175942020.

18 Daniel Selsam and Nikolaj Bjørner. Guiding high-performance sat solvers with unsat-core predictions. 11628 LNCS:336–353, 2019. doi:10.1007/978-3-030-24258-9_24.

19 Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L. Dill. Learning a SAT solver from single-bit supervision. In *Proceedings of ICLR 2019*. OpenReview.net, 2019. URL: https://openreview.net/forum?id=HJMC_iA5tm.

20 Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *JMLR*, 15:1929–1958, 06 2014.

21 Grigori S Tseitin. On the complexity of derivation in propositional calculus. In *Automation of reasoning*, pages 466–483. Springer, 1983.

22 Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Satzilla: Portfolio-based algorithm selection for SAT. *JAIR*, 32:565–606, 2008.