Deep Learning for Algorithm Portfolios

Andrea Loreggia
University of Padova
IBM Research, NY, U.S.A.
andrea.loreggia@gmail.com

Yuri Malitsky
IBM Research, NY, U.S.A.
yuri.malitsky@gmail.com

Horst Samulowitz
IBM Research, NY, U.S.A.
samulowitz@us.ibm.com

Vijay Saraswat
IBM Research, NY, U.S.A.
vijay@saraswat.org

Abstract

It is well established that in many scenarios there is no single solver that will provide optimal performance across a wide range of problem instances. Taking advantage of this observation, research into algorithm selection is designed to help identify the best approach for each problem at hand. This segregation is usually based on carefully constructed features, designed to quickly present the overall structure of the instance as a constant size numeric vector. Based on these features, a plethora of machine learning techniques can be utilized to predict the appropriate solver to execute, leading to significant improvements over relying solely on any one solver. However, being manually constructed, the creation of good features is an arduous task requiring a great deal of knowledge of the problem domain of interest. To alleviate this costly yet crucial step, this paper presents an automated methodology for producing an informative set of features utilizing a deep neural network. We show that the presented approach completely automates the algorithm selection pipeline and is able to achieve significantly better performance than a single best solver across multiple problem domains.

Introduction

Over the last decade, it has become an accepted fact that there is often no single approach that will dominate across a wide range of problem instances. Techniques like algorithm configuration (Hutter, Hoos, and Leyton-Brown 2011; Ansótegui, Sellmann, and Tierney 2009; Fitzgerald, Malitsky, and O'Sullivan 2015) can certainly improve the average performance of any parameterized solver for a particular dataset, such improvements are typically achieved by sacrificing some performance on a subset of instances. Therefore, if instead there are a number of highly specialized solvers available, techniques like algorithm selection are designed to automatically determine the most appropriate approach for any newly presented instance (Rice 1976). When applied to competitions in domains like SAT (Le Berre, Roussel, and Simon 2014), MaxSAT (Argelich et al. 2014), and CSP (van Dongen et al. 2009), it is commonly observed that a portfolio properly utilizing solvers from one or even two years ago can readily dominate over any new single solver. Developing new solvers is imperative and the only way to drive research forward, but it is clear that rather than being general purpose Jack-of-all-Trades programs, these new solvers need to be highly specialized for only a small subset of problems. It is then the job of algorithm selection techniques to identify when each of the solvers should be used.

Yet while there is now a plethora of competing algorithm selection approaches (Kotthoff 2014), all of them are fundamentally dependent on the quality of a set of structural features they use to distinguish amongst the instances. If the features are too noisy or uninformative, no selection technique will be able to make intelligent decisions. Over the years, each domain has defined and refined its own set of features, yet at their core they are mostly a collection of everything that was considered useful in the past. It is typically the job of filtering techniques to identify the most representative feature set for each selection technique for a particular set of solvers. In recent years, there have been a few attempts to augment this shotgun generation of features through the use of a systematic analysis of the latent features based on solver performances (Malitsky and O'Sullivan 2014), but in practice such approaches take a considerable amount of expertise to generate.

The focus of this paper is therefore to eliminate the human element from the feature generation process by using a deep learning approach. The paper notes that for a majority of domains, the specifics of any problem instance is typically expressed as a text document. For example, a SAT problem is typically represented in the DIMACS CNF format (Trick et al. 1993) where after a header, each line in the file describes the literals in each clause. Similarly, CSP problems can be represented in the XCSP format (Roussel and Lecoutre 2009) while MIPs can be represented in LP or MPS formats (ILOG 2006). Regardless of the format, the problem instances are represented in a text file. This paper, therefore presents a way to automatically convert any such text file into a grayscale square image, which can in turn be used to train a deep neural network to predict the best solver for the instance.

The proposed technique was applied across multiple datasets in SAT and CSP domains. We first benchmark the datasets by showing the performance of a state-of-the-art algorithm selection strategy, CSHC (Malitsky et al. 2013), uti-

lizing the established set of features. We subsequently evaluate the performance of the deep neural network to predict the best solver, as well as the quality of the output vector of such a network to act as a new feature vector for an existing selection strategy. Both of the new automated techniques are considerably better than the best single solver in any of the benchmarks, and while not quite at the level of the state-of-the-art portfolio on features studied for over a decade, our fully automated approach is shown to be competitive.

To the best of our knowledge this is the first tentative usage of deep learning in this area. Very recently introduced approaches share the idea of extracting knowledge from raw data without employing any crafted information. For example, a novel approach shows how a deep neural network can learn the semantic of simple arithmetic operations, such as addition and subtraction, by simply training the network using images. The input consists of two images of numbers while the output corresponds to the results of the selected arithmetic operation (Hoshen and Peleg 2015). The concept of number and arithmetic operator is left to be learnt by the neural network. This work can be thought of as a computer vision task of frame prediction and can be grouped with other approaches in the same area such as (Vinyals et al. 2014). Of a particular interest is another recent method that applies a temporal convolutional networks to text understanding where the applied technique has no knowledge of words, phrases or sentences nor any know-how about syntax or semantic structure (Zhang and LeCun 2015). Yet results evidence surprising achievement, even starting with only the simple character level input.

These techniques, however, are specialized to their perspective domains whereas the approach presented here aims to work across multiple problem representations, each with a unique grammar and internal organization of data. In short, a SAT file looks nothing like an XCSP file. Furthermore, even the problems themselves can vary dramatically in terms of their size, while most machine learning approaches rely on constant sized feature vectors. This problem can be abstracted away in text mining by relying on word counts or other relations, but in our domain, we might care about more than how frequently variable x_i appears. Therefore, the work in this paper is a proposed first step in developing a methodology of representing a diverse class of problem domains in a finite representation.

Algorithm Selection

Algorithm selection is the study of choosing the most appropriate solver for the problem at hand based on a descriptive set of features that describe the instance. In 2007, the SAT community was rocked by the introduction of a new solver that completely dominated their annual competition (Xu et al. 2009). SATzilla was a a simple portfolio that relied on ridge regression to predict the expected log-runtime of its constituent solvers, executing the one with the lowest expected runtime. Yet this straightforward application dramatically altered how we approach solver development. In the subsequent years, increasingly better portfolios came to the scene. ISAC (Kadioglu et al. 2010) utilized clustering to differentiate the instances, CPHydra (O'Mahony et al. 2008) and 3S (Kadioglu et al. 2011) then used scheduling to find the best sequence of solvers to evaluate. In 2012, SATzilla came out with a new method utilizing a random forest, reclaiming first place prizes in the SAT community (Le Berre, Roussel, and Simon 2014). Now the range of techniques is growing at a rapid rate and for an up to date list of related research, we refer the reader to the constantly updated survey (Kotthoff 2014).

Currently, based on the latest results in the SAT and MaxSAT competitions, a state-of-the-art portfolio is CSHC (Malitsky et al. 2013), which is the technique we employ in this paper. An acronym for cost-sensitive hierarchical clustering, CSHC bases its decision by training a forest of random trees. Each tree in the forest is trained with a splitting criteria that ensures that it divides the training data such that the instances in each child node, maximally agree on the solver they prefer. The partitioning stops when the amount of data in a child node becomes too small. To ensure variation of the forest, each tree is trained on a random subset of 70% of the data and a random subset of considered features. Therefore, whenever a new instance is presented, each tree votes for the best solver based on its data.

From Text to Images to Selection

There are many file specifications utilized, each one specifically formulated to most conveniently represent a particular problem. Satisfiability instances are typically represented in cnf format:

```
c SAT EXAMPLE
p cnf 3 2
1 -3 0
2 3 -1 0
```

In this representation a line beginning with a "c" is considered a comment, while "p" signals the problem type, number of variables and number of clauses. The subsequent lines list the literals belonging to each clause, with a minus sign depicting a negation. In the example above, the mathematical SAT problem represented is: $(x_1 \vee \neg x_3) \wedge (x_2 \vee x_3 \vee \neg x_1)$.

The CNF format efficiently captures all the necessary details about the SAT problem, but at the same time is unable to capture the requirements of a full constraint satisfaction problem. For something like that, the XCSP format is a better fit, which uses xml to first specify the domains, then the variables, and then the relations between the variables. The example below defines a problem with variables A_1 and A_2 that can take a value 1 or 2 and must each must be different.

```
<domains nbDomains="1">
 <domain name="d0" nbValues="2">1..2</domain>
</domains>
<variables nbVariables="2">
 <variable name="A1" domain="d0"/>
 <variable name="A2" domain="d0"/>
</variables>
<constraint name="c0" arity="2"
 scope="A1 A2"
 reference="global:alldifferent"/>
```

Certainly the SAT problem could be represented as a CSP. Albeit not always practical, one could even encode any NP complete problem into any other NP complete problem in polynomial time. But for our approach we want to be able to take potentially any problem definition and encode into something usable by a machine learning approach like a deep neural network. Therefore, this section shows how to take the above presented formats and convert them to gray-scale images with a predefined dimension n. We subsequently show how these images can be used to train and test a deep neural network using 10-fold-cross validation. In our specification, the output of the network represents a scoring of all solvers on the provided input instances, which can either be used directly as a selection approach, or as new features to be used by existing selection strategies.

Image generation

Converting text documents into images of a fixed size is a well studied topic for identification and prediction, but is not readily applicable to our scenario. For one, note that the vocabulary and grammar is vastly different between the CNF and XCSP formats, preventing us from taking any advantage of known structures. We particularly avoid such structures because we want our approach to be as general as possible. We similarly cannot take advantage of the typical approach of counting the frequency of words, since in most cases we do not care how frequently words appear together but the relations of which words they appear next to. Finally, problems can be of widely different sizes, with SAT instances ranging from hundreds to millions of clauses and CSP instance consisting of just dozens of lines. Ultimately the core issue is that we need a way to represent

The employed conversion process works as follows: For each available instance, the plain text file is read character by character and replaced with its corresponding ASCII code. Each such code is stored in a vector of length N, where N is the number of total characters in the input file. After reading the entire file, the vector is reshaped using the new dimension $\sqrt{N}$. We can now draw a square gray scale image for each instance using the ASCII code value for a shade of gray. For example, the following snippet of a CNF file:

```
88 1134 1972 0
699 81 -1082 0
-239 -1863 1594 0
```

is represented as the following vector of ASCII codes: `[56,56,32,49,49,51,52,32,49,57,55,...]`. Note that all characters are mapped – including spaces and line break symbols. Since ASCII codes range between 0 and 255 they can be mapped conveniently to gray scale. q

While this initial image representation is loss-free since there exists a one-to-one mapping from the original text to the image and vice versa, we now rescale the image to a predefined size (e.g., 128x128) using standard image scaling operators. Hence the process produces a set of images which are all of the same size. This is one key point of this work: we strongly believe that instances expose structure and self-similarity (Ansótegui et al. 2014) (patterns can easily been visualized using images) and that these properties can be maintained once we rescale the images. The images can be useful to visualize and to analyze these structures, regardless of the considered domain. While scaling the images incurs a high loss in information it seems to be the case that the retained structure is sufficient to address decision problems such as algorithm selection. Figures 1 and 2 show an image extracted from a SAT and CSP instance: patterns are easily visible in each image.

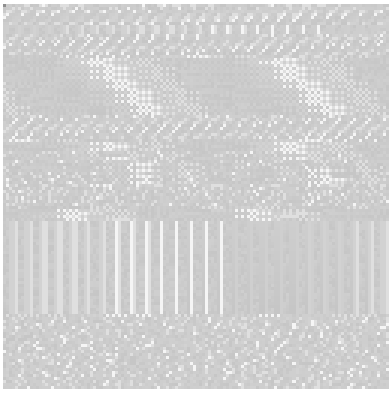

Figure 1: Image extracted from SAT problem instance.

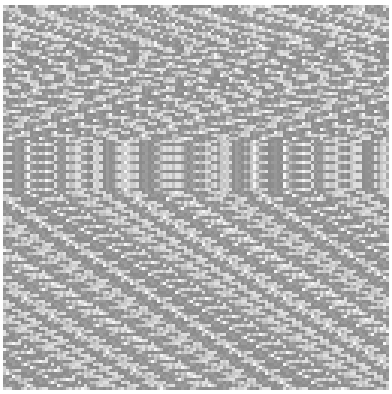

Figure 2: Image extracted from CSP problem instance.

Neural network

In machine learning, a neural network is a structure especially used for classification or regression tasks when the high dimensionality and non-linearity of the data make these tasks hard to accomplish. In the realm of visual data the standard is to employ convolutional neural networks (CNN). CNNs are directly inspired by the hierarchy of the cells in visual neuroscience (Hubel and Wiesel 1962). The same structure roughly resembles the one in the visual cortex (Felleman and Essen 1991). Nowadays it represents the state-of-the-art in image classification area (Krizhevsky, Sutskever, and Hinton 2012) and in many others such as speech recognition (Sainath et al. 2013) and face recognition (Taigman et al. 2014). Convolutional neural networks are specifically designed to deal with multi-dimensions input such as images.

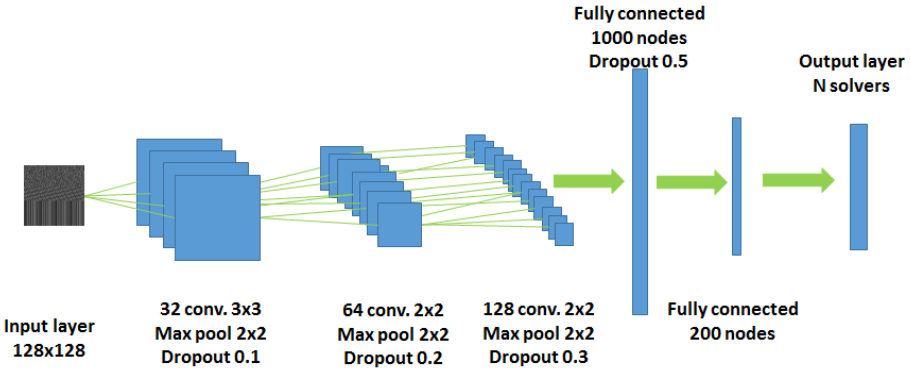

Figure 3: Deep convolutional neural network structure.

The modular approach of the deep network and the use of convolutional layers allow the early stage of the network to search for junctions of features while the use of pooling layers try to merge them semantically. The great success of convolutional neural networks on image related tasks inspired our approach in the context of algorithm portfolios: could an image representation of textual information leverage the capabilities of CNNs to perform algorithm selection? While the last section described how we converted textual representations to images, we describe the employed CNN model.

Our deep CNN network starts with three convolutional layers, each one followed by a max-pool layer and a dropout layer. At the very end of the network there are two fully-connected layers with a single dropout layer in the middle. The output layer is composed of m nodes, where m is the number of solvers. Dropout layers help to prevent overfitting (Srivastava et al. 2014) and make the neural network performances more stable in combination with other techniques, such as adjusting momentum and learning rate during the training phase.

The network uses the stochastic gradient descent (SGD) algorithm to speed-up the back-propagation and during the training phase it is updated using Nesterov momentum (Sutskever et al. 2013). The minibatch size is set to 128, learning rate is initially set to 0.03 and momentum is initially set to 0.9. Both are adjusted during the training phase with step size of 0.003 for learning rate and 0.001 for momentum. The non-linearity used is the rectify function $\phi(x) = max(0, x)$, while the output layer uses the sigmoid function $\phi(x) = 1/(1 + e^{-x})$.

Figure 3 represents the structure of the convolutional neural network implemented and used in all experiments. The figure also reports the number of filters used and their dimensions as well as the dimension for each convolutional layers and the probabilities used by the dropout layers and max-pool layers. Before training the neural network, the data is preprocessed in the following way: For each feature we subtract the mean and normalize each feature to have a standard deviation equal to 1. This preprocessing step has been shown to be beneficial for efficiency and performances of neural networks (LeCun et al. 1998).

The plain classification task (i.e. training the neural network to predict which is the best solver to use for a given instance) has resulted in poor performances. Consequently, we moved to a slightly easier binary regression task, which corresponds to train the neural network to predict whether a solver can solve the given instance or not. The objective loss function is the binary cross entropy:

$$L = -t \log(p) - (1 - t) \log(1 - p)$$

where $t \in \{0, 1\}$ is the ground truth value, and $p \in [0, 1]$ is the predicted value.

Discussion As desired the proposed approach is oblivious to any domain specific properties since its parsing problem instances character by character and does not rely on any given predefined structure. In general, the process exhibits very little bias — except for the step that scales the initial image to its miniature sized version. Our method relies on the fact that the employed scaling function (e.g., the default image scaling algorithm) retains the structure that is needed to perform algorithm selection. Ideally one would want to *learn* this reduction function so that the needed structure is retained without depending on a somewhat arbitrary transformation. In future work one could consider employing, for example, an LSTM (Hochreiter and Schmidhuber 1997) to learn the appropriate transformation function as well.

Experiments

We implemented the neural network described in Section using Python 2.7 and Lasagne 0.1. Lasagne is a framework based on Theano 0.7 (Bastien et al. 2012; Bergstra et al. 2010) which allows development of CNNs at a rather abstract and transparent level. In addition the framework allows exploitation of high performance GPUs. The main idea of Lasagne is to compose the neural network using different available layers stacking them on top of each other. For each layer it is possible to alter various parameters based on the layer's type. This rather straightforwardly leads to the implementation of the deep neural network. We have limited the training of the neural network to 100 epochs, since increasing the number of epochs made the neural network overfit due to the limited amount of data available in those domains. We have also experimented with different image sizes (e.g., 32x32 and 256x256) and while for domains with the highest

Dataset	CSHC	BSS	CNN	New Feat.
Industrial	95.6 ± 2.56	92.6 ± 2.50	93.0 ± 2.73	93.7 ± 2.56
Random	96.5 ± 1.89	77.4 ± 1.74	90.7 ± 1.90	90.1 ± 1.45
Crafted	91.6 ± 2.45	77.2 ± 4.75	78.2 ± 6.32	79.9 ± 4.11
CSP	94.5 ± 2.94	65.4 ± 4.98	82.1 ± 2.31	78.2 ± 2.97

Table 1: Percentage of solved instances.

number of instances (e.g., random) larger images resulted in better performances, we decided to choose 128x128 which seemed to have the best trade-off between number of input parameters and performance.

We empirically evaluated this novel approach using different data sets coming from the satisfiability (SAT) and constraint programming domains (CSP). The SAT datasets are publicly available on the SAT competition website and are usually divided into the following three sub-domains: industrial, random and crafted. We have the performances of 29 solvers for each of about 800 instances for industrial, more than 2,200 instances of random and about 730 of crafted. We also use the performances of 22 solvers for each of the almost 1,500 instances in the CSP domain. The CSP instances come from the CSP Competition (csp 2009) and include non-trivial instances from problem classes such as Timetabling, Frequency Assignment, Job-Shop, Open-Shop, Quasi-group, Costas Array, Golomb Ruler, Latin Square, All Interval Series, Balanced Incomplete Block Design, and many others. This set includes both small and large arity constraints and all of the global constraints used during the CSP solver competitions: all-different, element, weighted sum, and cumulative.

For each dataset we performed the prediction task using a 10-fold cross validation approach. Hence, we first split a dataset into a training and test set. The train set is then split further into train and validation splits using a ratio of $75\%/25\%$. The neural network was trained using the images corresponding to the instances of a given dataset. The trained neural network was then used to predict which solver could finish a given test instance within the timelimit. The neural network outputs for each solver a value between 0 and 1, where 0 indicates that the solver cannot finish the given instance and 1 means the opposite. For evaluation, we select the solver whose output obtains the highest value. This strategy will be referred to as CNN. Alternatively, instead of relying solely on the neural network to make the correct decision on which solver to use, it is also possible to interpret the output layer as a new feature vector. A specialized approach for algorithm selection (e.g., (Xu et al. 2012; Malitsky et al. 2013)) can then be used to try to refine the selection process. We refer to this approach as "New Feat" in the results that follow.

The results obtained by our methods are compared with the ones obtained using regular manually crafted features with CSHC (Malitsky et al. 2013), which represents a state-of-the-art approach in the area of algorithm portfolios. We use our own implementation of the classifier.

Similar to a majority class in a plain classification task the baseline in this setting is the following: after executing all solvers on the train dataset and computing the average running time elapsed by each one one chooses as prediction the algorithm that behaves on average the best. This selection strategy we label the Best Single Solver (BSS).

Dataset	CSHC	BSS	CNN	New Feat.	VBS
Industrial	691	921	903	831	463
Random	939	1,775	1,143	1,163	741
Crafted	1,803	2,598	2,242	2,219	1,544
CSP	489	1,501	914	1,022	301

Table 2: Average running time in seconds for various selection strategies.

Tables 1 and 2 summarize our empirical results. While our approach is not able to achieve state-of-the-art performance, it does give better performances than the baseline on all considered domains. Note that this is without relying on features crafted by expert humans.

In particular, Table 1 shows the percentage of solved instances of a given domain using one of the before-mentioned methods. The presented deviations are based on the statistics after performing 10-fold cross validation. We believe that the performance could be boosted further if more problem instances would be available for training. While the performance on random and CSP can be clearly distiniguished from the baseline, the difference in performance on the benchmarks with a smaller set of available instances (industrial and crafted) is not as pronounced.

We also consider how our approach performs in terms of average run time per instance. To this end we compared the prediction from the neural network not only in terms of number of solved instances but also in terms of average runtime used by the prediction to solve the instance. Table 2 reports these results. As before the first column reports the results obtained with CSHC. The very last column "VBS" corresponds to the oracle performance which corresponds to the average runtime that one would achieve if for each instance one would always select the fastest solver. Once again, this new approach is performing better then the BSS in all of the scenarios and the gap to the state-of-the-art is within reasonable limits for a fully automated approach.

Table 3 reports the number of misclassifications that the neural network incurred on SAT instances. As already said, the very last layer of the neural network is a vector of values in $[0, 1]$, where 0 in position i means that the i-th solver cannot solve the given instance, 1 otherwise. For any test instance and for any solver, we know the actual ability of the solver (e.g., solve or not solve). So given a test instance, we counted how many output values of the neural network are

Dataset	Misclassification
Industrial	4.63
Random	4.93
Crafted	10.84

Table 3: Average number of misclassified solvers per instance by the convolutional neural network.

Dataset	BSS	Random
Industrial	92.6 ± 2.50	62.1 ± 4.28
Random	77.4 ± 1.74	31.6 ± 1.96
Crafted	77.2 ± 4.75	55.1 ± 9.53

Table 4: Average percentage of solved instances for an agent that chose randomly.

wrong by changing to 0 all the values less than equal to 0.5 and changing to 1 the others, comparing the outcomes with reality. On the industrial and random benchmarks the neural network makes very few errors where less than 5 out of 29 solvers are predicted incorrectly per instance. The number of misclassifications grows to about 10 for the crafted dataset. But note that as long as the CNN selects a solver that can solve the instance the resulting portfolio will still yield good performance. In addition existing algorithm selection techniques can further learn patterns on top of the predictions made by the neural network to automatically correct mistakes and improve overall predictions.

Given the presented results and considering the complexity of our approach there are a number of alternate hypothesis to which the rise in performance can be attributed to. For one, it is possible to imagine that if the majority of the solvers in the portfolio are reasonable, than simply randomly guessing the best solver could achieve comparable behavior. Table 4 shows that this is not the case in the examples we presented. The random agent always performs worse than the best single solver and its executions is far from the ones of the neural network, suggesting that the neural network is able to extract some structure from the data that enables it to select the correct solver.

An alternate explanation for the performance can be that our neural network is not really learning to differentiate between instances, but between the generators used to create them. This is possible since certain random generators can put larger clauses towards the top of the instance, order clauses lexicographically, or subconsciously incorporate some other kind of order. It is well know in the literature that neural networks are notorious at picking up on these external patterns, producing misleading results. To test this hypothesis, we tried repeating the experiments with the SAT instances but this time randomly shuffling the clauses of the instances and ordering the variables in each clause lexicographically. However, the performance of the resulting portfolios was still comparable to the performance presented in Table 1.

An interesting question is if combining the manually crafted and automatically generated features results in further improvements in terms of performance. To that end we simply append the original and new features and use the composed feature vector to train CSHC. However, the achieved performance based on the combined feature vector does not result in any improvements over just using the original features. We have not yet explored if techniques like feature selection applied to the combined feature vector would boost performance.

Overall, the results seem to strongly suggest that the introduced approach of converting textual representations of problem instances into gray scale images does capture some structure of the underlying instance. This structure can then be picked up and exploited with the right tools. The result means that we can completely remove human expertise from the picture of algorithm selection, allowing the tools to be readily applied to new fields.

Conclusion

Algorithm selection has drastically changed the practice of research of algorithms. The significant amount of research has shown that instead of creating Jack-of-all-Trades methodologies, we should instead focus on highly specialized techniques, selecting the best strategy for the problem at hand. However, in order to get selection techniques to work to their full potential a substantial amount of human expertise is required to create the features necessary to differentiate between instances. In this work we take the first step to fully automating this process in hopes to make algorithm selection an easier tool for researchers to use off-the shelf.

We introduced the usage of deep learning techniques in automated algorithm portfolios by training a neural network using images extracted from problem instances. We show how this new approach gives solid performances on different domains, even though we have not used any domain knowledge. Avoiding feature generation and the usage of domain knowledge makes this new approach very appealing on a variety of different domains.

While the presented approach continuously out performed any single solver, it still naturally lags behind carefully configured and specialized approaches. Nonetheless, there are many subsequent lines of research to pursue. The effects of introducing some domain knowledge to filter out repetitive irrelevant words. Alternatively, another way of encoding words rather than character by character could be possible. Furthermore, learning how to compress the initial image in order to retain the necessary structure could not only improve performance further, but would also remove the bias induced by the employed scaling function. Finally, could one leverage existing knowledge representations such as the one available from Imagenet (Deng et al. 2009) to improve performance? All these directions are now being considered for future work.

References

Ansótegui, C.; Bonet, M. L.; Girldez-Cru, J.; and Levy, J. 2014. The fractal dimension of sat formulas. In Demri, S.; Kapur, D.; and Weidenbach, C., eds., *IJCAR*, volume 8562 of *Lecture Notes in Computer Science*, 107–121. Springer.

Ansótegui, C.; Sellmann, M.; and Tierney, K. 2009. A gender-based genetic algorithm for the automatic configuration of algorithms. In *Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming*, CP'09, 142–157. Berlin, Heidelberg: Springer-Verlag.

Argelich, J.; Li, C. M.; Manyà, F.; and Planes, J. 2014. Max-SAT Evaluation 2014.

Bastien, F.; Lamblin, P.; Pascanu, R.; Bergstra, J.; Goodfellow, I. J.; Bergeron, A.; Bouchard, N.; Warde-Farley, D.; and Bengio, Y. 2012. Theano: new features and speed improvements. *CoRR* abs/1211.5590.

Bergstra, J.; Breuleux, O.; Bastien, F.; Lamblin, P.; Pascanu, R.; Desjardins, G.; Turian, J.; Warde-Farley, D.; and Bengio, Y. 2010. Theano: A cpu and gpu math compiler in python. In van der Walt, S., and Millman, J., eds., *Proceedings of the 9th Python in Science Conference*, 3 – 10.

2009. CSP Solver Competition Benchmarks. `http://www.cril.univ-artois.fr/~lecoutre/benchmarks.html`.

Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Fei-Fei, L. 2009. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*.

Felleman, D. J., and Essen, D. C. V. 1991. Distributed hierarchical processing in the primate cerebral cortex. *Cerebral Cortex* 1:1–47.

Fitzgerald, T.; Malitsky, Y.; and O'Sullivan, B. 2015. Reactr: Realtime algorithm configuration through tournament rankings. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence*, IJCAI15.

Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. In *Neural Computation 9 (8)*, 17351780.

Hoshen, Y., and Peleg, S. 2015. Visual learning of arithmetic operations. *CoRR* abs/1506.02264.

Hubel, D., and Wiesel, T. 1962. Receptive fields, binocular interaction, and functional architecture in the cat's visual cortex. *Journal of Physiology* 160:106–154.

Hutter, F.; Hoos, H.; and Leyton-Brown, K. 2011. Sequential model-based optimization for general algorithm configuration. In *Learning and Intelligent Optimization - 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers*, 507–523.

ILOG. 2006. Cplex 10.0 file formats.

Kadioglu, S.; Malitsky, Y.; Sellmann, M.; and Tierney, K. 2010. Isac - instance-specific algorithm configuration. In Coelho, H.; Studer, R.; and Wooldridge, M., eds., *ECAI*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, 751–756. IOS Press.

Kadioglu, S.; Malitsky, Y.; Sabharwal, A.; Samulowitz, H.; and Sellmann, M. 2011. Algorithm selection and scheduling. In *International Conference on Constraint Programming*, volume 6876, 454–469.

Kotthoff, L. 2014. Algorithm selection for combinatorial search problems: A survey. *AI Magazine* 35(3):48–60.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In Bartlett, P. L.; Pereira, F. C. N.; Burges, C. J. C.; Bottou, L.; and Weinberger, K. Q., eds., *NIPS*, 1106–1114.

Le Berre, D.; Roussel, O.; and Simon, L. 2014. SAT 2014 competition.

LeCun, Y.; Bottou, L.; Orr, G.; and Muller, K. 1998. Efficient backprop. In Orr, G., and K., M., eds., *Neural Networks: Tricks of the trade*. Springer.

Malitsky, Y., and O'Sullivan, B. 2014. Latent features for algorithm selection. In *The Seventh Annual Symposium on Combinatorial Search (SOCS)*.

Malitsky, Y.; Sabharwal, A.; Samulowitz, H.; and Sellmann, M. 2013. Algorithm portfolios based on cost-sensitive hierarchical clustering. In *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*.

O'Mahony, E.; Hebrard, E.; Holland, A.; Nugent, C.; and O'Sullivan, B. 2008. Using case-based reasoning in an algorithm portfolio for constraint solving. In *Proceedings of the 19th Irish Conference on Artificial Intelligence and Cognitive Science*.

Rice, J. 1976. The algorithm selection problem. *Advances in Computers* 15:65–118.

Roussel, O., and Lecoutre, C. 2009. XML representation of constraint networks: Format XCSP 2.1. *CoRR* abs/0902.2362.

Sainath, T. N.; rahman Mohamed, A.; Kingsbury, B.; and Ramabhadran, B. 2013. Deep convolutional neural networks for lvcsr. In *ICASSP*, 8614–8618. IEEE.

Srivastava, N.; Hinton, G. E.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research* 15(1):1929–1958.

Sutskever, I.; Martens, J.; Dahl, G. E.; and Hinton, G. E. 2013. On the importance of initialization and momentum in deep learning. In *ICML (3)*, volume 28 of *JMLR Proceedings*, 1139–1147. JMLR.org.

Taigman, Y.; Yang, M.; Ranzato, M.; and Wolf, L. 2014. Deepface: Closing the gap to human-level performance in face verification. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Trick, M.; Chvatal, V.; Cook, B.; Johnson, D.; McGeoch, C.; and Tarjan, B. 1993. The second dimacs implementation challenge.

van Dongen, M.; Lecoutre, C.; Manyà, F.; and Planes, J. 2009. CSP Evaluation 2009.

Vinyals, O.; Toshev, A.; Bengio, S.; and Erhan, D. 2014. Show and tell: A neural image caption generator. cite arxiv:1411.4555.

Xu, L.; Hutter, F.; Hoos, H.; and Leyton-Brown, K. 2009. Satzilla2009: an automatic algorithm portfolio for sat. solver description. SAT Competition.

Xu, L.; Hutter, F.; Shen, J.; Hoos, H.; and Leyton-Brown, K. 2012. SATzilla2012: Improved algorithm selection based on cost-sensitive classification models. Solver description, SAT Challenge 2012.

Zhang, X., and LeCun, Y. 2015. Text understanding from scratch. *CoRR* abs/1502.01710.