Visual Sudoku Puzzle Classification: A Suite of Collective Neuro-Symbolic Tasks

Eriq Augustine[1], Connor Pryor[1], Charles Dickens[1], Jay Pujara[2], William Yang Wang[3] and Lise Getoor[1]

[1]University of California, Santa Cruz
[2]University of Southern California
[3]University of California, Santa Barbara

Abstract

Neuro-symbolic computing (NeSy) is an emerging field that has the goal of integrating the low-level representational power of deep neural networks with high-level symbolic reasoning. Due to the youth of the field and the complexity of neuro-symbolic integration, there are few benchmarks that showcase the powers of NeSy, and even fewer built specifically with NeSy in mind. To address the lack of NeSy benchmarks, we introduce Visual Sudoku Puzzle Classification (ViSudo-PC). ViSudo-PC is a new NeSy benchmark dataset combining visual perception with relational constraints. The goal of the benchmark is to both highlight opportunities and elicit challenges. In addition to providing a new NeSy benchmark suite, we also provide an exploratory analysis that showcases ViSudo-PC's difficulty and possibilities.

Keywords

Benchmark, Dataset, Neuro-Symbolic Integration, Relational Data, Structured Prediction

1. Introduction

Integrating neural and symbolic reasoning is a long-standing challenge in the machine learning community. Neuro-symbolic computing (NeSy), which combines low-level neural perception and logic-based reasoning [1, 2, 3], is a promising area of research that aims to integrate these concepts in a seamless fashion. NeSy systems have shown the advantage of incorporating neural and logical reasoning, including the ability to learn with less data, robustness to noise, the ability to perform joint reasoning (structured prediction), and more. Unfortunately, there is a dearth of NeSy datasets that are complex, relational,

Figure 1: A ViSudo-PC puzzle with MNIST digits. Guidelines added for exposition.

NeSy 2022, 16th International Workshop on Neural-Symbolic Learning and Reasoning, Cumberland Lodge, Windsor, UK
eaugusti@ucsc.edu (E. Augustine); cfpryor@ucsc.edu (C. Pryor); cadickens@ucsc.edu (C. Dickens); jpujara@ucs.edu (J. Pujara); william@cs.ucsb.edu (W. Y. Wang); getoor@ucsc.edu (L. Getoor)

and realistic. A comprehensive NeSy test suite designed with the ability to vary the structural constraints and perceptual difficulty is a challenge facing the community.

There are a variety of tasks and datasets commonly used in NeSy research. Many involve visual reasoning. Examples include identifying the subject or context of the image such as Visual Relationship Detection [4], Semantic Image Interpretation [5], and Visual Genome [6]. These datasets support interesting and complex visual tasks. However, the complex nature of the task can lead to ambiguous answers (e.g., even humans may misunderstand the context of an arbitrary image). Additional NeSy test domains involve reasoning with knowledge graphs such as FB15k and WN18 [7]. These datasets often lack direct subsymbolic information, requiring that information to be generated from other sources (e.g., from word embeddings). Finally, MNIST Addition [8] is another popular NeSy dataset. MNIST Addition uses MNIST digit images as operands in addition equations, with the goal of predicting the sum of the digit images. MNIST Addition is an excellent NeSy testbed, however, it is limited by the ease of MNIST image classification. With MNIST classifiers that can achieve over 99.9% accuracy [9], MNIST Addition (both single and multi variants) is possible using little symbolic reasoning.

Our goal is to create a comprehensive NeSy benchmark that can be used to further NeSy research. Such a benchmark needs to take into account the nature of neuro-symbolic computing at each step. Specifically, a benchmark made for the NeSy community should: 1) include self-contained symbolic and subsymbolic information, both of which are necessary to solve the problem; 2) contain settings/tasks with varying degrees of hardness; 3) include entities that can be collectively reasoned over; and 4) have unambiguous labels.

In this work, we introduce a novel benchmark specifically designed for NeSy systems, Visual Sudoku Puzzle Classification (ViSudo-PC). ViSudo-PC expands on the concept of visual Sudoku puzzles introduced in Wang et al. (2019) and visual Sudoku puzzle classification introduced in Pryor et al. (2022). Given a Sudoku puzzle constructed from images as input, the classification task is to determine whether the Sudoku puzzle is correctly solved. Preforming well on the classification of visual Sudoku puzzles requires systems that are able to reason about the perceptual information in the images as well as the collective information from Sudoku constraints. ViSudo-PC expands upon the perceptual challenge of previous MNIST compositional tasks [8, 10, 11] by drawing images from four different sources. Additionally, ViSudo-PC includes a collection of progressively harder tasks.

Our key contributions are as follows: 1) We construct ViSudo-PC, an extensive NeSy benchmark that integrates four canonical visual datasets into five tasks of varying difficulty requiring symbolic and sub-symbolic reasoning to solve, 2) We perform an exploratory evaluation over two ViSudo-PC tasks to quantify the difficulty of these tasks in different settings, and 3) We discuss ways that the data and tasks of ViSudo-PC can be extended and improved.

2. Benchmark Data

Visual Sudoku Puzzle Classification (ViSudo-PC) expands upon the classification task proposed in [11]. The data includes completed Sudoku puzzles, along with their classification as “correct"

(solved) or “incorrect"[1][2]. For those unfamiliar with Sudoku, Sudoku is a puzzle game in which there is a 9x9 grid, called a “puzzle" or “board", in which each cell is populated with numbers 1 – 9. A puzzle is correct if no row, column, or non-overlapping 3x3 subgrid (or “block") contains the same number. Classifying whether a Sudoku puzzle is solved correctly simply involves checking the three types of constraints. Following [10], rather than providing symbolic information (e.g., labels) for the cell content, we can complicate the problem by providing subsymbolic information in the form of images. The images can be of digits (or, as we'll see later, other objects) for each cell as in Figure 1. Note that no information is provided about the label of each cell, so a classification system must learn to identify or distinguish between the cell labels at the same time as to check if the puzzle is solved.

2.1. Data Sources

We build upon existing image classification work [12, 13, 14, 15] to create Sudoku puzzles where the cell contents and labels originate from multiple data sources. Each data source provides 28 pixel by 28 pixel grayscale images covering a different domain of objects, examples of these images are displayed in Figure 9. Table 1 summarizes the data sources used by ViSudo-PC to construct Sudoku puzzles.

Dataset	Train Examples	Test Examples	Labels	Subject
MNIST	60,000	10,000	10	Digits
EMNIST-ML	697,932	116,323	47	English Letters
FMNIST	60,000	10,000	10	Fashion Items
KMNIST	60,000	10,000	10	Japanese Characters

Table 1: An overview of the different data sources available in ViSudo-PC.

MNIST MNIST is one of the most well-known and widely used image classification datasets [12]. It is composed of 70,000 examples of handwritten digits distributed roughly evenly across the ten digit classes.

Extended MNIST (EMNIST) extends MNIST by introducing additional examples of handwritten digits as well as examples of handwritten English letters [13]. EMNIST provides several different ways to group/classify the data, including by author, by class (arranged into 62 classes denoting [0-9], [a-z], and [A-Z]), and by merged classes. The merged classes setting combines similar uppercase and lowercase letters, e.g., “c" & “C", “m" & “M”, and “o" & “O”, into 47 total classes. ViSudo-PC uses the merged classes, but removes digits to avoid overlap with the MNIST digits. We refer to this subset of EMNIST as *EMNIST Merged Letters* (EMNIST-ML). The images provided in EMNIST are transposed horizontally and rotated 90 degrees anti-clockwise. To maintain consistency with the other data sources, each EMNIST-ML image is adjusted to an upright position.

[1]Data is available at https://linqs-data.soe.ucsc.edu/public/datasets/ViSudo-PC/v01/.
[2]ViSudo-PC also provides a data generation tool discussed in Appendix B.

Fashion-MNIST (FMNIST) uses images of fashion products [14] instead of digits. Classes include items such as “Coats", “Bags", and “Trousers". Samples of each FMNIST class are illustrated in Figure 9c. The complex nature of fashion items and wide range of variants makes classification FMNIST images considerably harder than standard MNIST [14].

Kuzushiji-MNIST (KMNIST) uses Kuzushiji Japanese characters [15]. Kuzushiji is a cursive form of Japanese writing rarely used today. To reduce the 49 character Japanese alphabet down to 10 classes, KMNIST chooses one character from each of the 10 Hiragana rows (representing different consonant sounds). The stylistic nature of a cursive writing variant, like Kuzushiji, makes KMNIST intrinsically more difficult than standard MNIST.

2.2. Puzzle Construction

ViSudo-PC puzzles are square and can be any dimension d with an integer square root, as long as enough cell labels are provided by the data sources (e.g., MNIST, with 10 classes, alone can only support Sudoku puzzles with a dimension of 9 or less). To construct ViSudo-PC puzzles, cell labels are first selected from the relevant data sources. The exact method of choosing data sources and cell labels varies between tasks and is discussed in detail in Section 3. Cell labels are randomly selected for each cell, ensuring that no Sudoku constraint is violated. Once cell labels are determined, images of those labels are assigned to each cell. To create a pool of images for each cell label, train and test splits from each data source are merged, shuffled, partitioned by label, and split into train, test, and validation image pools. Images are never shared between splits.

To create negative puzzle examples (incorrectly solved puzzles), existing correct puzzles are corrupted. Our method of corruption allows ViSudo-PC to contain incorrect puzzles that are just slightly incorrect, instead of incorrect puzzles that are constructed by random and would likely have many mistakes. Puzzles are corrupted in one of two ways: via *replacement* or via *substitution*. Replacement corruptions randomly choose a location in a puzzle and an alternate label, and then replaces that cell with an image uniformly sampled from the split’s pool of images for that label. Substitution corruptions swap two random cells in the same puzzle. After each corruption is made, a coin with a configurable bias is flipped to see if another corruption of the same type is performed. Finally, each corrupted puzzle is checked to ensure that multiple corruptions did not create a correct puzzle.

To increase the connectivity of puzzles, ViSudo-PC allows for the introduction of *overlap* into the data. Overlap is when the same image is used multiple times when generating puzzles. Adding overlap gives the predictor an opportunity to recognize the same entity being used in the same or different puzzles. A predictor employing joint reasoning can take advantage of this opportunity to improve performance.

The degree of overlap is controlled by a parameter ω. From a collection of images I, $\omega|I|$ images are uniformly sampled with replacement. The sampled images are added to I, which is then shuffled, forming a new collection of $|I| + \omega|I|$ images.

3. Benchmark Tasks

Five different tasks are provided as a part of the ViSudo-PC benchmark, each providing increasingly difficult problems. In all settings, the goal is to classify Sudoku puzzles as correct or incorrect. Cell labels are provided for debugging, but should never be used in any official task.

Basic In this task, a single data source is specified and the first d cell labels from the data source are used, where d is the dimension of the puzzles. These are the cell labels for the puzzles, and then images for the cell labels are chosen randomly. This extends the original Visual-Sudoku-Classification problem from Pryor et al. (2022) by including alternate data sources.

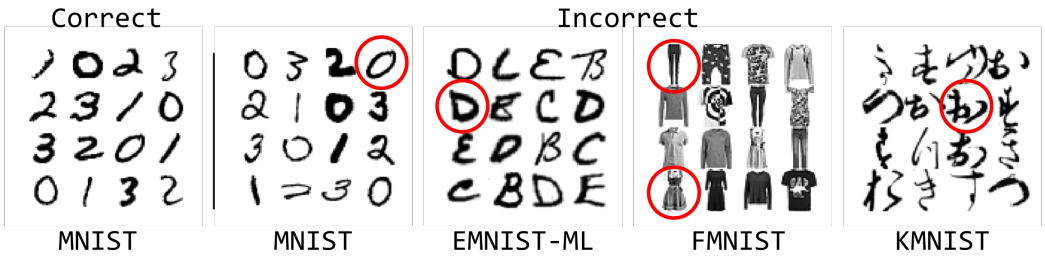


Figure 2: Examples of a correct (left) and four incorrect (right) puzzle instances of the Basic task. Mistakes are highlighted with red circles.

Random Label Per Split (PerSplit) This task builds upon the Basic task by randomizing the cell labels used in each split. PerSplit randomly selects d cell labels from the specified data sources to use throughout all train, test, and validation puzzles. Any non-empty subset of data sources can be specified. The challenge posed by PerSplit is that any model/architecture used must be effective on several types of images, and not specific to one label set. For example, an architecture specialized to classify MNIST digits may fail to classify the shoes and purses in FMNIST. Any architecture that performs well on all variations of this task must be able to deal with the digits, English letters, clothes, and Japanese characters that may appear in a single puzzle.

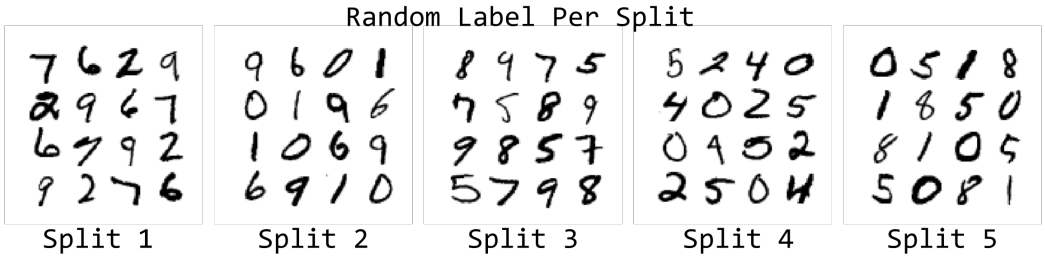


Figure 3: Examples of correct puzzle instances from five different splits generated using the MNIST dataset on the Random Label Per Split task. Note the different set of labels used in each split.

Random Label Per Puzzle (PerPuzzle) This task increases the difficulty by re-sampling the d cell labels for each individual puzzle. So for each puzzle, a set of d cell labels is uniformly

sampled from the specified data sources. Note that this task becomes considerably more difficult when more data sources are used, since the pool of possible cell labels is larger. All cell labels present in the test and validation puzzles are guaranteed to be used in train puzzles. For this task (and the following tasks), methods that rely solely on subsymbolic information (image pixels) will likely have great difficulty. Because of the larger pool of cell labels, each cell label may be represented by fewer examples than in previous tasks. To perform well in this task, models may need to distinguish between cell labels for each puzzle and use the symbolic information in Sudoku constraints, instead of learning an image classifier on the train split.

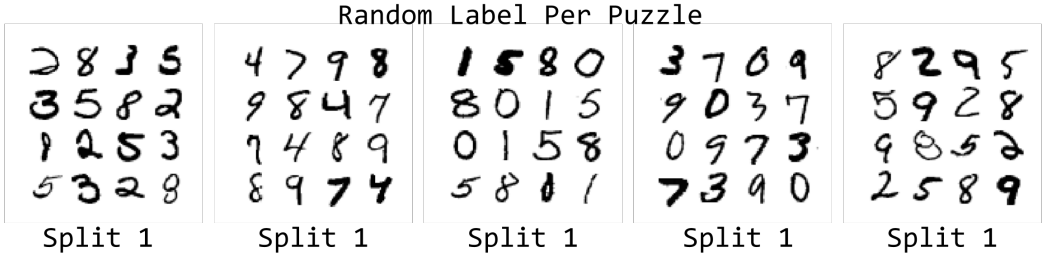


Figure 4: Examples of correct puzzle instances from a single split generated using the MNIST dataset on the Random Label Per Puzzle task. Note the different set of labels used for each puzzle within the same split.

Random Label Per Cell (PerCell) Instead of limiting each puzzle to using d cell labels, this task randomly chooses a cell label for each cell in each puzzle. Thus it can use as many as d^2 cell labels (limited by the puzzle size and provided data sources). The number of cell labels used per puzzle is randomly chosen and that information is not provided to the predictor outside of the train puzzles. This is the only task that violates the full rules of Sudoku, as more that d cell labels are potentially used. In this task, a puzzle is considered correct as long as the row, column, and block constraints of Sudoku are not violated, i.e., no duplicate cell labels appear in any row, column, or block. Again, we guarantee that all cell labels present in the test and validation puzzles are also present in train puzzles. The potentially large and unknown number of cell labels makes this task extremely challenging for any system that relies on an image classifier. To perform well on this task, a predictor needs to be able to discriminate between cell labels without seeing many examples of each and without knowledge of the number of cell labels.

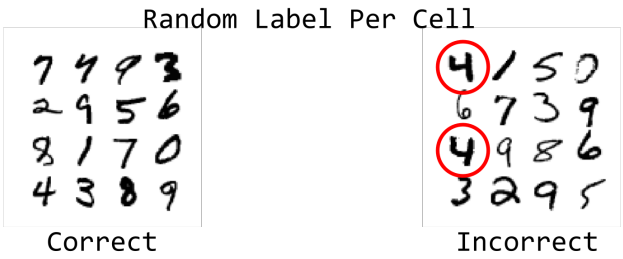


Figure 5: Examples of a correct (left) and incorrect (right) puzzle instances generated using the MNIST dataset on the Random Label Per Cell task. Mistakes are highlighted with red circles.

Transfer The final task is a transfer learning task. The same process used for Basic is used here, except that two disjoint sets of cell labels are chosen, one for training and another one for test and validation. For example, when MNIST is used as a data source with $d = 4$, the cell labels [0 - 3] are present in the train set, while [4 - 7] are present in the test and validation sets.

Figure 6: Examples of correct puzzle instances from the train and test portions of a single split generated using the MNIST dataset on the Transfer task. Note the disjoint labels used in the train and test portions of the same split.

4. Exploratory Evaluation

We provide an initial exploratory evaluation of ViSudo-PC using the Basic and Random Label Per Split tasks. We investigate the following questions: 1) Is there a difference in the difficulty of each data source? 2) How does the use of multiple data sources affect performance? 3) How does overlap affect performance?

4.1. Models

We evaluate over three models from Pryor et al. (2022) using hyperparameters specified in the paper; all unspecified parameters were left at their default values.

Baseline-Digit This model takes as input the cell labels of a Sudoku puzzle and outputs a probability of the puzzle being valid. Note that this can be seen as the best possible scenario (or cheating), where the neural model is able to correctly identify every cell image. This model uses a feedforward multi-layer perceptron trained to minimize the cross-entropy loss. Formally, this neural baseline consists of 3 fully connected dense layers of sizes 16, 512, and 256 each with a ReLu activation and a final dense output layer of size 1 with a softmax activation.

Baseline-Visual This model takes as input the pixels for each cell in a Sudoku puzzle and outputs the probability of the puzzle being valid. This model uses a convolutional neural network multi-layer perceptron trained to minimize the cross-entropy loss. Formally, this neural baseline consists of 3 convolutional layers with kernel size of 3, where each is followed by a max pooling layer of size 2 with stride 2. This then feeds into the same model as bld.

NeuPSL A NeSy model that has distinct neural perception and symbolic reasoning components. The NeuPSL neural model takes as input the pixels for each cell in a Sudoku puzzle, and outputs

a probability distribution for each class, which it then feeds into a symbolic model that verifies the Sudoku constraints. Formally, the NeuPSL neural model is a simple image classifier first mentioned in Manhaeve et al. (2021). This neural model consists of 2 convolutional layers with kernel size of 5, where each is followed by a max pooling layer of size 2 with stride 2. This then feeds into fully connected dense layers of sizes 256, 120, 84 with a ReLu activation and a final dense output layer of size n with a softmax activation (where n is the number of classes). The symbolic PSL model implements the rules of Sudoku as described in Pryor et al. (2022), i.e., has no duplicate digits in any row, column, or square.

4.2. Data Source Difficulty

To assess the difficulty of each data source for the tasks presented by ViSudo-PC, we first look at the difficulty of each task in the simpler context of image classification. Table 2 shows the state-of-the-art image classification performance for each data source at the time of writing [17]. All data sources achieve accuracy in the 90s, with MNIST and KMNIST performing the best and both achieving more than 99% accuracy, while EMNIST is the hardest with an accuracy of only 91.59%.

By examining each data source's image classification performance, we can get an idea of the best possible performance a naive ViSudo-PC model can achieve. Where a naive model simply attempts to classify each cell in a puzzle independently (assuming cell labels are supplied to the model). ViSudo-PC provides both 4x4 and 9x9 puzzles (with the ability to generate larger puzzles), therefore the expected accuracy of a naive model is $(ImageAccuracy)^{16}$ and $(ImageAccuracy)^{81}$ respectively[3].

Data Source	Model	Image Accuracy	Expected 4x4 Accuracy	Expected 9x9 Accuracy
MNIST	CNN Ensemble [9]	99.91%	98.57%	98.57%
EMNIST-Merged[4]	WaveMix [18]	91.59%	24.52%	00.08%
FMNIST	DARTS [19]	96.91%	60.52%	07.87%
KMNIST	SpinalNet [20]	99.15%	87.23%	50.09%

Table 2: State-of-the-art image classification accuracy on all data sources [17], as well as each model's expected performance on 4x4 and 9x9 ViSudo-PC puzzles.

Additionally, we assess the performance of the three models on the Basic task using 50 4x4 training puzzles from each data source. Table 3 shows the results of the three models on each data source. The Baseline-Visual is unable to generalize over any of the data sources. Baseline-Digit, however, performs approximately the same over all data sources, as it does not use any perceptual information. Unsurprisingly, NeuPSL performs the best on MNIST, which has the simplest image cell labels. And despite being the most difficult data source for the state-of-the-art image classifiers, NeuPSL performed second best on EMNIST-ML.

[3]The expected accuracy only includes performance on positive examples, and additionally excludes cases where multiple classification mistakes are made which result is an accidental correct classification.
[4]Digits are included in this setting, but excluded for ViSudo-PC.

Data Source	Baseline-Digit	Baseline-Visual	NeuPSL
MNIST	0.71 ± 0.04	0.51 ± 0.02	0.88 ± 0.02
EMNIST-ML	0.69 ± 0.05	0.50 ± 0.01	0.79 ± 0.09
FMNIST	0.70 ± 0.04	0.50 ± 0.03	0.74 ± 0.04
KMNIST	0.71 ± 0.03	0.50 ± 0.02	0.65 ± 0.12

Table 3: Performance of the CNN baselines and NeuPSL on the Basic task using 50 4x4 training puzzles from each data source. Mean area under the receiver operating characteristic curve (AuROC) along with standard deviation over 10 splits is reported.

4.3. Performance across Multiple Data Sources

To examine the impact of using multiple data sources for puzzle generation, we ran the Random Label Per Split task using 4x4 puzzles generated with data from one or more data sources. As shown by Figure 7, NeuPSL performs well in almost all settings, outperforming Baseline-Digit, but struggling more whenever KMNIST is used. This result is consistent with NeuPSL's previous performance with KMNIST. Baseline-Digit has very consistent performance and is almost agnostic to the different data sources. As expected, Baseline-Visual fails to generalize and produces consistently poor performance.

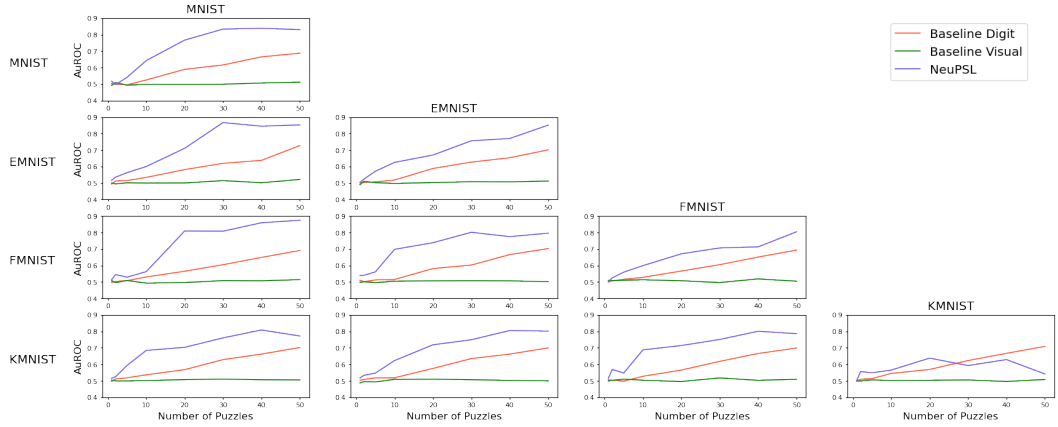


Figure 7: Mean area under the receiver operating characteristic curve (AuROC) of the CNN baselines and NeuPSL over ten splits on the Random Label Per Split task using different combinations of data sources.

4.4. Overlap Performance

To determine the effect of overlap on performance, we ran the three models on the Basic task using differing amounts of overlapping images in 4x4 puzzles from each data source. As shown in Figure 8, both the Baseline-Digit and NeuPSL models show a benefit from increasing the amount of overlap. NeuPSL, which is able to collectively reason, shows much larger improvements as the amount of overlap is increased than Baseline-Digit, which may just be benefiting from fewer

unique images. Here, Baseline-Visual also fails to generalize and cannot beat random guessing.

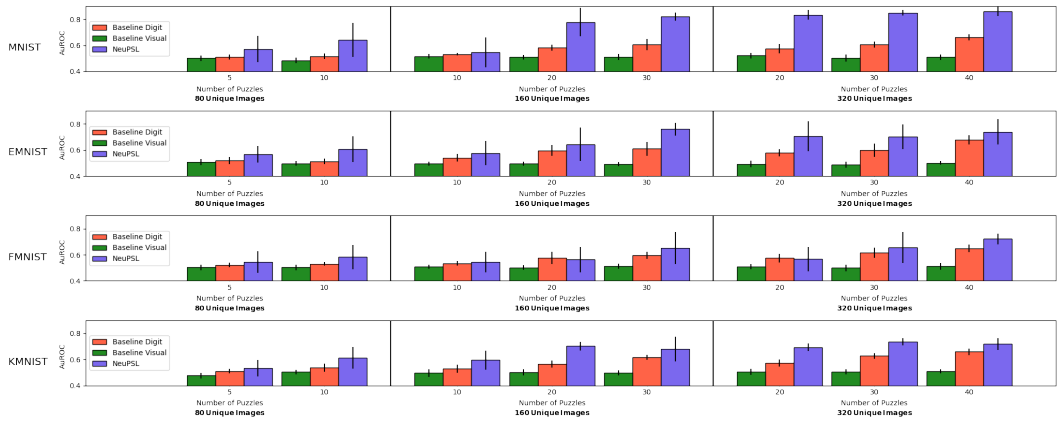


Figure 8: Mean area under the receiver operating characteristic curve (AuROC) and standard deviation of the CNN baselines and NeuPSL over ten splits on the Basic task using a differing amount of overlapping images in 4x4 puzzles from each data source.

5. Discussion

There are several interesting ways in which both the data and tasks in ViSudo-PC can be extended by including additional structural information. Cell labels can be expanded to hierarchies of labels, e.g., a MNIST zero may be a part of the cell label hierarchy: 0 → Digit → Alphanumeric → Glyph. These hierarchies can then be used to create a variety of new tasks, at different abstraction levels. For example, one could define a task where cell labels sharing a common hierarchical ancestor are considered the same. Another possible new task involves adding additional constraints on the Sudoku problem. For example, requiring specific blocks to contain cells from different label classes, e.g., all numbers or letters. Recall that in the current set of tasks, no cell labels are given, and results are not evaluated over cell labels, only over puzzle classification. Another set of new tasks can be introduced by including cell labels in either the inference or evaluation process. Finally, an additional interesting direction is introducing confounding information. For example, following [21], confounding information in the form of color could be explicitly added to the data generation process.

Acknowledgments

This work was partially supported by the National Science Foundation grants CCF-1740850 and CCF-2023495.

References

[1] T. R. Besold, A. S. d'Avila Garcez, S. Bader, H. Bowman, P. M. Domingos, P. Hitzler, K. Kühnberger, L. C. Lamb, D. Lowd, P. M. V. Lima, L. de Penning, G. Pinkas, H. Poon, G. Zaverucha, Neural-symbolic learning and reasoning: A survey and interpretation, arXiv (2017).

[2] A. S. d'Avila Garcez, M. Gori, L. C. Lamb, L. Serafini, M. Spranger, S. N. Tran, Neural-symbolic computing: An effective methodology for principled integration of machine learning and reasoning, Journal of Applied Logics 6 (2019) 611–632.

[3] L. De Raedt, S. Dumančić, R. Manhaeve, G. Marra, From statistical relational to neuro-symbolic artificial intelligence, in: IJCAI, 2020, pp. 4943–4950.

[4] C. Lu, R. Krishna, M. Bernstein, L. Fei-Fei, Visual relationship detection with language priors, in: European Conference on Computer Vision (ECCV), 2016, pp. 852–869.

[5] I. Donadello, L. Serafini, A. D. Garcez, Logic tensor networks for semantic image interpretation, arXiv (2017).

[6] R. Krishna, Y. Zhu, O. Groth, J. Johnson, K. Hata, J. Kravitz, S. Chen, Y. Kalantidis, L.-J. Li, D. A. Shamma, et al., Visual genome: Connecting language and vision using crowdsourced dense image annotations, International Journal of Computer Vision (IJCV) 123 (2017) 32–73.

[7] A. Bordes, N. Usunier, A. Garcia-Durán, J. Weston, O. Yakhnenko, Translating embeddings for modeling multi-relational data, in: International Conference on Neural Information Processing Systems (NeurIPS), 2013, pp. 2787–2795.

[8] R. Manhaeve, S. Dumancic, A. Kimmig, T. Demeester, L. De Raedt, Deepproblog: Neural probabilistic logic programming, in: NeurIPS, 2018, pp. 3753–3763.

[9] S. An, M. Lee, S. Park, H. Yang, J. So, An ensemble of simple convolutional neural network models for mnist digit recognition, arXiv (2020).

[10] P.-W. Wang, P. Donti, B. Wilder, Z. Kolter, Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver, in: International Conference on Machine Learning (ICML), 2019, pp. 6545–6554.

[11] C. Pryor, C. Dickens, E. Augustine, A. Albalak, W. Wang, L. Getoor, Neupsl: Neural probabilistic soft logic, arXiv (2022).

[12] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE 86 (1998) 2278–2324.

[13] G. Cohen, S. Afshar, J. Tapson, A. V. Schaik, Emnist: Extending mnist to handwritten letters, in: International Joint Conference on Neural Networks (IJCNN), 2017, pp. 2921–2926.

[14] H. Xiao, K. Rasul, R. Vollgraf, Fashion-mnist: A novel image dataset for benchmarking machine learning algorithms, arXiv (2017).

[15] T. Clanuwat, M. Bober-Irizar, A. Kitamoto, A. Lamb, K. Yamamoto, D. Ha, Deep learning for classical japanese literature, arXiv (2018).

[16] R. Manhaeve, S. Dumančić, A. Kimmig, T. Demeester, L. De Raedt, Neural probabilistic logic programming in deepproblog, Artificial Intelligence 298 (2021) 103504.

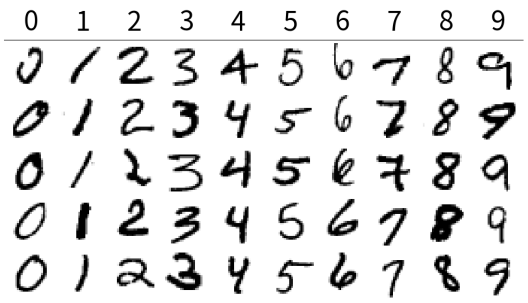
[17] P. W. Code, Image classification, https://paperswithcode.com/task/image-classification, 2022. Accessed: 2022-05-27.

[18] P. Jeevan, A. Sethi, Wavemix: Resource-efficient token mixing for images, arXiv (2022).

[19] M. S. Tanveer, M. U. K. Khan, C.-M. Kyung, Fine-tuning darts for image classification, in: International Conference on Pattern Recognition (ICPR), 2021, pp. 4789–4796.
[20] H. Kabir, M. Abdar, S. M. J. Jalali, A. Khosravi, A. F. Atiya, S. Nahavandi, D. Srinivasan, Spinalnet: Deep neural network with gradual input, arXiv (2020).
[21] B. Kim, H. Kim, K. Kim, S. Kim, J. Kim, Learning not to learn: Training deep neural networks with biased data, in: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019, pp. 9012–9020.

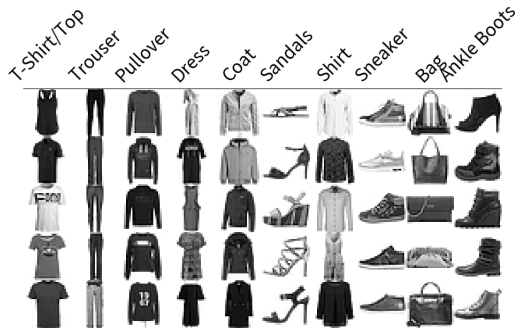
A. Data Sources

Examples of each data source are provided in Figure 9. MNIST, FMNIST, and KMNIST have all their provided classes displayed. EMNIST-ML is shown with 10 of its 47 classes.

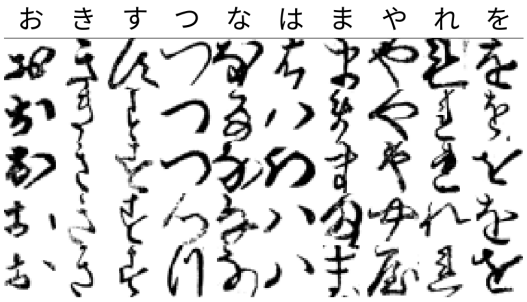


(a) MNIST

(b) EMNIST-ML



(c) FMNIST



(d) KMNIST

Figure 9: Examples of each data source. The top row of each data source indicates the true label for each column.

B. Data Generation

ViSudo-PC provides a data generation tool that allows users to construct their own ViSudo-PC datasets[5]. The settings in Table 4 are used to create the data provided with ViSudo-PC[6]. Here, we provide a brief description of each parameter.

Parameter	Value
Dimension	$\{4, 9\}$
Data Sources	$PowerSet(\{\text{MNIST}, \text{EMNIST} - \text{ML}, \text{FMNIST}, \text{KMNIST}\}) - \{\}$
Train Count	$\{1, 2, 5, 10, 20, 30, 40, 50, 100\}$
Test Count	100
Valid Count	100
Overlap	$\{0.0, 0.5, 1.0, 2.0\}$
Corrupt Chance	0.5

Table 4: Data settings used in the provided ViSudo-PC data.

Dimension Dimension determines the size of the Sudoku puzzles generated. All Sudoku puzzles in ViSudo-PC are square, and the dimension d is the number of cells on each side of the puzzle. Additionally, because Sudoku puzzles require square blocks within each puzzle, the puzzle dimension must have an integer square root.

Data Sources Data sources determines the cell labels and images. For many of the tasks discussed in Section 3, data may come from more than one source.

Train/Test/Valid Counts The number of correct puzzles to generate for each split. During the corruption process, the same number of incorrect puzzles will also be generated.

Overlap The constant ω controls the number of examples that are duplicated, as discussed in Section 2.2.

Corruption Chance While generating negative instances as described earlier, the chance of continuing the corruption process after each corruption is made. This controls how many mistakes are in the negative examples.

[5]Code is available at https://github.com/linqs/visual-sudoku-puzzle-classification.
[6]Data is available at https://linqs-data.soe.ucsc.edu/public/datasets/ViSudo-PC/v01/.

C. Benchmark Size

Table 5 shows the sizes of NeSy datasets built from MNIST-style images. ViSudo-PC sizes include all task and data source configurations with a train size of 100 and an overlap constant (ω) of 0.0. Parameters controlling ViSudo-PC sizes are discussed in Appendix B.

Name	Data Sources	Size Variant	Images / Instance	Cell Labels	Unique Images	Instances
Visual Sudoku [10]	{ MNIST }	9×9	81	10	60,000	9,000
MNIST-Addition [16]	{ MNIST }	$1 + 1$	2	10	60,000	30,000
		$2 + 2$	4	10	60,000	15,000
		$3 + 3$	6	10	60,000	7,500
		$4 + 4$	8	10	60,000	3,750
VSPC [11]	{ MNIST }	4×4	16	10	60,000	2,000
ViSudo-PC	{ MNIST,EMNIST-ML, FMNIST,KMNIST }	4×4	16	77	877,932	106,000
		9×9	81	77	877,932	106,000

Table 5: Size of NeSy datasets composed of MNIST-style images. The *Size Variant* column describes the dataset variant used, e.g., a $4x4$ ViSudo-PC puzzle or an MNIST-Addition with single digit numbers ($1 + 1$).