# Neural Analogical Reasoning

Atharv Sonwane[1,*,†], Abhinav Lalwani[1,*], Sweta Mahajan[2], Gautam Shroff[3] and Lovekesh Vig[3]

[1]APPCAIR, BITS Pilani, K K Birla Goa Campus
[2]Indian Institute of Technology, Delhi
[3]TCS Research, New Delhi

**Abstract**

Symbolic systems excel at reusing and composing modular functional units when solving problems such as simple analogical reasoning. However, they are less amenable to processing real-world data (e.g. images), and rely on additional (often hard-coded) mechanisms to convert such high-dimensional data to symbolic descriptions. In this work, we describe a modular approach 'Neural Analogical Reasoning' wherein elementary neural transformations operate and compose on distributed representations of high-dimensional inputs. We apply this approach on a class of visual analogical reasoning problems that involve discovering the sequence of transformations by which pairs of input-output images are related, so as to analogously transform future inputs. This can be viewed as a program synthesis task and solved via symbolic search if represented in symbolic form. Instead, we search for a sequence of elementary neural network transformations that manipulate distributed representations of the inputs. We present two variations of learning useful representations for this task and compare both with end-to-end meta-learning based approaches to demonstrate the importance of performing an explicit search.

**Keywords**

neural reasoning, visual analogy making

## 1. Introduction

Consider the class of visual reasoning problems as demonstrated in Figure 1 in which each task involves a *functional analogy* (i.e., $x : f(x) :: y : f(y)$) [1]. Analogical reasoning in this context can be thought of as the reuse of some transformation $f$ and analogy making as the discovery of these transformations. By modelling these transformations as compositions of elementary functional units, we can cast analogy making into a program synthesis problem. Given a set of input-output pairs $S$ and a set of functional units $T$, we must discover a composition $f = T_1.T_2.....T_n$ where $T_i \in T$ such that $f(x_i) = x_o$ for all $(x_i, x_o) \in S$. This program $f$ can then applied to a query image to generate an *analogous* output. The key questions to be answered in such a frameworks are how these elementary functional units $T_i$ are to be represented and how they can be ensured to be composable with each other and usable across a wide enough range of inputs.

*NeSy 2022, 16th International Workshop on Neural-Symbolic Learning and Reasoning, Cumberland Lodge, Windsor, UK*
*These authors contributed equally.
†Corresponding author.
f20181021@goa.bits-pilani.ac.in (A. Sonwane)

Classical reasoning systems can solve such problems via discrete search over compositions of elementary primitives [2, 3]. There have also been advances in However, they require both the inputs and the elementary primitives to be represented with predefined symbols. For the problem in Figure 1, the images would need to be explicitly annotated with discrete symbolic features such as the type of shape and its location in the image. Such use of predefined symbols allows classical reasoning systems to leverage compositionality and reuse computational units to perform robust reasoning. However, it restricts their applicability in more complex domains where feature extraction may be non-trivial while also making it difficult to discover new knowledge from data.

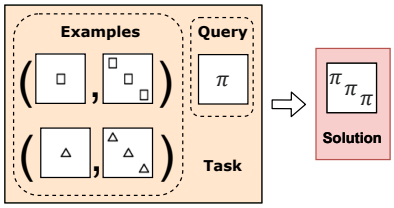



Figure 1: Given example input-output image pairs, generate the analogous output for the given query image.

Deep Neural Networks (DNNs), on the other hand, have shown an impressive ability to learn compressed distributed representations of high-dimensional data in complex domains such as images [4], text [5] and graphs [6]. Despite these successes, it remains unclear how to reason over such learned representations to solve analogy problems. In this paper, we take the approach of combining a discrete search procedure with neural primitives that manipulate distributed representations in a latent space. This allows us to leverage both the flexibility of DNNs and the ability of discrete search to construct arbitrarily complex compositions.

Our proposed system consists of (1) a distributed representation space $Z$ into and from which images can be converted with an encoder $E : I \to Z$ and decoder $D : Z \to I$ and (2) neural primitives $T_i : Z \to Z$ that approximate the action of symbolic primitives within this representation space and that are composable with each other. Given a task as described in Figure 1, our system searches over sequences of such neural primitives to find a program that transforms the given input to the given output. By using a search procedure in place of a completely end-to-end learned system, we retain the flexibility of having independent functional units that can be reused, debugged and composed arbitrarily to build transformations in an interpret able manner. And using learned primitives means tedious hand-crafting of new symbols and procedures can be replaced with providing relevant examples of the transformation to be learned. Thus, our system aims to retain the power of compositional reasoning that symbolic primitives provide while being able to generalize beyond the practical capacity of a purely symbolic approach to richer inputs.

In such a framework, the ability of primitives to compose with each other while generalising to similar but unseen domains is essential for successful analogical reasoning. We obtain almost arbitrarily composable primitives by using neural networks that manipulate distributed representations within the same latent space ($T_i : Z \to Z$). To learn these neural primitives as well as the latent space they operate on, we take inspiration from the Neural Algorithmic Reasoning framework [7], which describes how to *neuralize* an erstwhile symbolic algorithm (in our case, transformation) so that it can deal with rich inputs directly without requiring them to be explicitly transformed to a symbolic form. We also describe and evaluate two variations of learning distributed representations of the input - a *guided* approach in which representation space is derived from symbolic descriptions of the inputs and an *unguided* approach) in which the representation is learned directly from the high-dimensional inputs (in this case images).

**Contributions** Through this paper we (1) introduce a challenging few-shot visual analogical reasoning task; (2) propose a *neural reasoning* framework that makes use of learned primitives along with an explicit search procedure to solve this task; (3) evaluate two different methods for representation learning within this framework; (4) demonstrate that elementary primitives learned through this approach are able to both compose and generalise - satisfying essential requirements of analogical reasoning; (5) show the importance of a search procedure when solving the task by comparing with neural networks based end-to-end meta-learning approaches.

## 2. Related Works

Analogical reasoning has been considered a fundamental trait of human intelligence for the last half-century of AI research [8]. More recently, there has been a focus on characterising similar types of abstract reasoning as the fundamental building blocks of intelligence [9, 10, 11]. We refer to [1] for a review of various broad approaches to analogical making in AI systems, with our particular method building *functional analogies*. However, the ability to represent concepts in a modular way and to transfer them across domains still eludes many modern machine learning approaches [12]. Recent attempts at utilising neural networks for reasoning tasks focus on creating modular architectures of neural networks that can be learned end to end [13, 14, 10]. However, the reliance on fixed architectures limits the ability to learn and reuse functional units. Further, these approaches limit inference to applying the trained network once or at most a few times, rather than searching for the solution as is done in symbolic program synthesis [2, 3]. Our system is able to reuse neural transformation units in an unconstrained manner by exploiting the the ability of search to create arbitrarily complex compositions. This idea of inferring programs made up of neural modules has also been used for visual question and answering [15]. However this utilises a learned sequence model to generate programs and solves a classification task as opposed to a generative one.

The Neural Algorithmic Reasoning [7] framework takes a different approach to reasoning with neural networks. Instead of learning end-to-end on a particular task, it proposes to learn a symbolic algorithm as a neural network together with a high dimensional representation space on which it can act. Such a neural algorithm can then be utilised in various rich domains by learning an encoder and decoder for it. This approach has proven successful for learning to execute graph algorithms [16] and is even able to transfer knowledge to learn related graph algorithms [17]. We use a variant of this framework to construct neural networks that learn from input and output data to execute symbolic primitives over a learned representation space. We also investigate the properties of this representation space since learning disentangled representations has been shown to be an important for visual abstract reasoning [18, 19, 20].

Other symbolic approaches to analogical reasoning include learning to rank possible relations based on features from a relational database [21] as well as Structure Mapping [22] and more recent approaches [23] that use of graph representation learning to emulate Structure Mapping. Our method differs from such approaches by operating directly on images without requiring structured symbolic descriptions during test time. Our work also builds on a previous short abstract [24], specifically by exploring an *unguided* way of representation learning which enables the overall system to work without any symbolic descriptions at all.

## 3. Methodology

**Problem Setting** We introduce a class of visual analogical reasoning problems as shown in Figure 1 to motivate our 'neural reasoning' system. Each problem consists of input-output image pairs which are related by an unknown transformation. Given these pairs, the task is to discover the transformation and apply it to a query image to generate the solution. The images considered consist of items from a set of 20 simple shapes and handwritten characters (from the Omniglot dataset [11]) placed on a $3 \times 3$ grid. The transformations between the input and output images are programs composed of either control flow operations (`reset` and `out`) or spatial transformations of shapes in the image. The latter consist of positional shifts in the ordinal directions (e.g. `shift-right` moves each shape in the image one grid-position to the right), shape conversions (e.g. `to-square` converts each shape in the image to a square), and affine transformations (e.g `affine-right` skews the the shape to the right).

This problem setting can be interpreted as solving weakly supervised evaluation tasks which consists of a input-output image pairs and a query image for which we have to generate the output *given* some strongly supervised training data consisting of similar input-output pairs and the associated transformation procedure. We utilise the strongly supervised data to learn: (1) a representation space for the input data and (2) neural transforms that manipulate distributed representations in this space. These can then be used to solve the weakly supervised testing tasks.

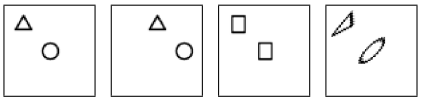



Figure 2: Example of applying various transformations to an image from the problem setting. **(a)** original image $x$, **(b)** `shift-right`(x), **(c)** `to-square`(x), **(d)** `affine-right`(x)

**Representation Learning** Reasoning requires constructing compositions of elementary functions. To use neural networks as functional units, we first need to obtain a latent space of real vectors into which we can embed the domains and co-domains of our original elementary functions. In particular, we are interested in representation spaces that are rich enough to represent a wide variety of shapes and sparse enough to represent concepts distinctly enough to allow for robust manipulation. We approach this in two related ways: (1) a *guided* representation derived from symbolic descriptions of the input and (2) an *unguided* representation learned directly from images (*unguided*). The *guided* approach generates more composable transforms at the cost of requiring annotated data and augmentations to enable generalisation The *unguided* approach generalises more naturally and can be trained without annotations but suffers from worse transform composability.

In the *guided* approach, we aim to shape the latent space to include information about the input images that is relevant to the transformations, namely shape and position. We first train an autoencoder (Encoder: $E_s$, Decoder: $D_s$) on symbolic descriptions of the input images present in the training data by minimising the negative log-likelihood loss between inputs and the reconstructed vectors. The symbolic descriptions used are concatenations of a one-hot vector denoting the shape present in the image and another denoting the position of this shape in the grid. We then train a CNN based encoder $E_x$ that takes images (denoted by $x$s) as inputs to fit to the outputs of $E_s$ (with frozen weights) by minimising the mean-squared-error between

the embeddings of images $E_x(x^{(i)})$ and embeddings of the corresponding symbolic vectors $E_s(s^{(i)})$: MSE $\left(E_x(x^{(i)}), E_s(s^{(i)})\right)$. To handle shapes at test time that were not seen during training, we also include an additional shape-label (`unseen`) and train $E_x$ to map shapes (whose labels weren't included in the symbolic description) to the latent vectors generated by $E_s$ corresponding to the `unseen` labels. While this approach ensures that the latent space encapsulates relevant information, it is limited by the reliance on symbolic descriptions. Addition of any new attributes of the input over which we would like to include transformations (e.g. a new shape, or continuous - such as affine - transformations of existing shapes) would require modifying the input space (size of one-hot input vector), requiring both latent space and transforms to be re-trained.

Learning the latent representation in an *unguided* way alleviates these limitations at the cost of not always capturing relevant information. In the *unguided* approach, we train a CNN based autoencoder $(E_x, D_x)$ by minimising mean-squared-error for reconstructions: MSE $\left(D_x(E_x(x^{(i)})), x^{(i)})\right)$. We expect the autoencoder to generalize to unseen shapes and positions given enough data. One of the main issues encountered here is the accumulation of noise after applying each transform, which degrades compositional performance. Using a sigmoid activation after the final layer of the decoder drastically reduces noise as the image pixels are binary. Further details can be found in the appendix attached.

**Transform Training** Our system makes use of neural networks to approximate the actions of the elementary spatial transformations that make up the mapping between input and output images. These *neural primitives* $T_i \in T$ need to (1) be universally composable with each other by operating on the same distributed representation space $Z$ (i.e. they must be of the form $T_i : Z \to Z$) and (2) generalise to latent representations of similar images enabling the transfer of analogical concepts (here compositions $f = T_1.T_2.\ldots.T_n$) across inputs. For each elementary transformation `tf` we construct a neural network $T_{\texttt{tf}}$ and train it to satisfy the constraint that $\texttt{tf}(a) = b \implies T_{\texttt{tf}}(E(a)) = E(b)$ for some encoder $E$, using the procedure from [24]. Further details can be found in the attached appendix.

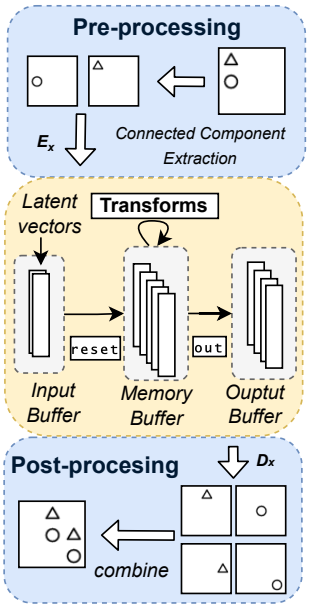


Figure 3: Program Execution

**Program Execution** We model mappings between input and output images in the visual analogy tasks as programs composed of primitives which are either control-flow operations (`reset` and `out`) or spatial transformations. Figure 3 describes the execution of such programs. Since spatial transformations act independently on each shape, we first extract individual shapes by identifying connected components within the image. Each extracted image is converted into a latent vector using encoder $E_x$ and stored in the input and memory buffers. The program is executed sequentially with neural primitives being applied to vectors in the memory buffer and control flow operations indicating transfer of vectors between buffers. After execution is complete, the vectors present in the

output buffer are converted to images using $D_x$ in the *unguided* case or by manual construction according to symbolic descriptions obtained using $D_s$ in the *guided* case. These are combined together to form the result. Further details can be found in [24] and the appendix.

**Searching for Solutions** Given a representation space and a set of transforms we can solve the analogy tasks being considered by searching over possible programs. A naive breadth first search has a branching factor 12 (4 shift operations, 4 change shape operations, 2 affine operations, `reset` and `out`), making it intractable for longer programs. Instead, we utilise the fact that both the input and output images are available, to extensively prune the search tree, details of which can be found in [24] and the attached appendix.

## 4. Results & Discussion

**Compositional Abilities of Transformations** A symbolic analogue to our approach would simply be a search over sequences of predefined hard-coded primitives. Our system uses learned primitives acting on a learned latent representation. Since these are an approximation to the underlying symbols, there may be errors in the system which lead to incorrect solutions or an inability to find a solution. To evaluate how well neural primitives in our system compose, we use of a dataset of input-output pairs, generated from 20000 random programs upto length 10 (details present in appendix), and compute the ratio of pairs for which a valid program is found.

The results are summarised in Figure 4. The *guided* approach is able to find the correct solution programs for 100% of the examples in the dataset. The *unguided* approach shows similar performance with an overall success rate of 94.7 %. The minor drop in performance of *unguided* can be attributed to (1) the absence of the symbolic annotations during training, (2) verification of the generated output is performed using MSE with the target, which is more susceptible to noise than comparing multi-hot vectors (used in *unguided*) and (3) the inclusion of the affine transforms resulting in noisy reconstructions for certain shapes since these more complex transforms are not learned as well as the others. Despite this, the high success rate of both approaches demonstrates that independently trained neural transforms can compose robustly enough to be used in our approach.

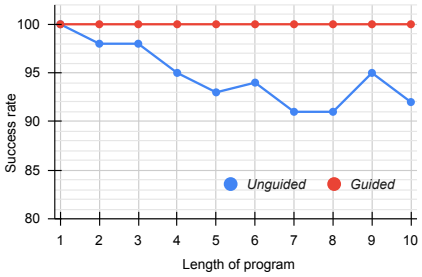
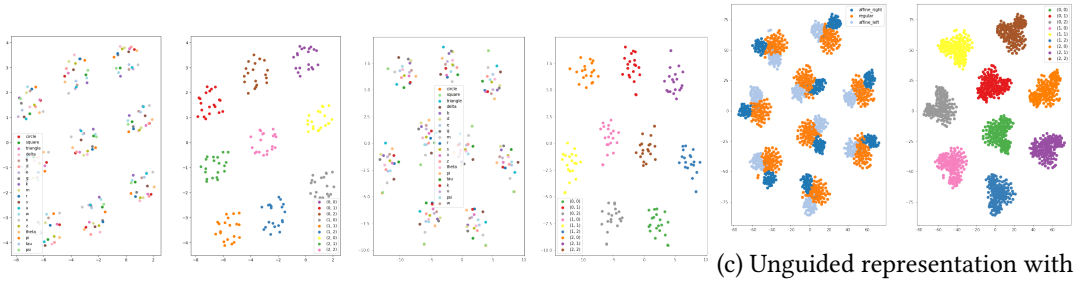



Figure 4: Performance at discovering programs of varying length.

**Comparison with End-to-end neural approaches** The analogical reasoning task presented in the paper can be thought of as a few-shot learning problem, where an output has to be found for a query given input-output examples - a setting well suited for end-to-end meta learning algorithms. In order to demonstrate the significance of discrete search in performing compositional reasoning, we compare our method against three end-to-end NN based meta-learning algorithms - Conditional Neural Processes (CNP) [25], Matching Networks (MAN) [26],

| Approach | Guided | Unguided | CNP | MAN | MAML |
|---|---|---|---|---|---|
| **Performance** | **100%** | **94.7%** | 64% | 74% | 73% |

**Table 1**
Performance of our approach (in bold) and comparison with fully NN based meta-learning algorithms (CNP, MAN, MAML) on a simplified few-shot classification version of the task.



(a) Guided representation

(b) Unguided representation

(c) Unguided representation with rotations and affine transformations included

**Figure 5:** Visualisation of latent representations of input images using t-SNE. Each point represents an embedding for a specific image consisting of a single shape in a grid position on the board. The plots on the left colour the points according to what shape is contained in the image. The plots on the right colour the points according the grid position the shape in the image.

Model-Agnostic Meta-Learning (MAML) [27] - on a few-shot classification version of the analogical reasoning task (details present in attached appendix). From the results in Table 1, we can see that despite being presented with a more difficult task (generation vs classification), our approach is able to significantly outperform end-to-end meta learning techniques.

**Representation Learning** The learned representation space upon which compositional reasoning is being performed is a critical part of our system. The experiments discussed above demonstrate that this space can be both disentangled enough to support robust transformations, as well as diverse enough to support interpolation and generalisation. To explore this further, we visualise the latent embeddings of images with a single shape in Figure 5. We can see that for both the *guided* and *unguided* cases, the representation space is strongly structured around the grid positions of the shapes in the images. For *unguided*, the clusters are more sparsely structured, indicating the presence of another component of variation that captures more shape information. Finally, Figure 5c shows that affine transformations of shapes are closer to each other than to the original shapes, indicating that `affine` transforms causes a shift in the distribution of latent embeddings. This explains why excluding affine transformations of seen shapes when training transforms causes reduced performance. Such transforms being applied to affine shapes would introduce significant noise since they are not aware of this distribution shift. This highlights the importance of having a well structured latent space.

**Generalisation** To be of practical use for computation and reasoning, our system needs to be able to generalise to visual analogy tasks with similar images but with characteristics not

| | Regular | Transforms | Representation |
|---|---|---|---|
| Guided | 100% | 95% | 100% |
| Unguided | 94.7% | 94.8% | 92.4% |

**Table 2**
End-to-end performance on generalisation experiments. The *Regular* case is where all 20 shapes are used for training and evaluation of both representations and transforms.

present in the training data. To evaluate this, we exclude images containing 5 (of the 20 total) shapes from the training data, but include these images during testing (performed similar to section 4). We evaluate generalisation for the transforms and representation space (encoder and decoder) separately. The results in Table 2 indicate that both the *guided* and *unguided* approaches are able to generalise well to shapes not present during training. This suggests that with a diverse and large enough training set, our representation space could be rich enough to support spaces with a large variety of inputs.

In the *unguided* approach, `shift` transforms work robustly for images containing shapes not seen during training, indicating that the representation space captures information about position in a disentangled manner. The `affine` transformations, which manipulate the shape itself, also work well for shapes *unseen* during transform training, but only if the affine versions of *seen* shapes are seen when learning the representation space. This is explained by the distribution shift in latent embeddings caused by `affine` transformations as seen in Figure 5c and discussed in section 4. Incorporating explicit measures to disentangle the distributed representations with respect to continuous shape transformations, in addition to shape vs position, may alleviate this need, enabling robust system-level generalization. More generally, these results suggest that representations learned directly and without supervision can capture relevant information in a distributed enough manner to support reasoning within our framework; and that providing explicit symbolic guidance aids in compositionality with some loss in generalizability.

**Conclusion** Our neural analogical reasoning approach applies compositional reasoning via search on transformations of distributed representations, both learned via neural networks, thus substituting handcrafting of symbols and procedures with provision of the relevant training data. Experiments show that our approach generalises to data drawn from related but different distributions (i.e., unseen shapes). Future work can tackle more complete representation learning methods to avoid extraction of connected components, enabling the system to tackle more complex problems with overlapping and arbitrarily sized shapes. We envisage a more general system for analogical reasoning (in the limited sense used here) based on our NAR approach: one that can operate on rich input distributions while continuously learning to detect when new shapes and/or transformations are encountered and retrain its elements (representation space and/or new transformation network(s)) appropriately.

## Acknowledgments

This work is supported by “The DataLab” agreement between BITS Pilani, K.K. Birla Goa Campus and TCS Research. We thank Tirtharaj Dash and Ashwin Srinivasan for their input.

## References


[1] H. Prade, G. Richard, Analogical proportions: why they are useful in ai, in: Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI 2021), Montreal, 2021, pp. 21–26.

[2] K. Ellis, C. Wong, M. Nye, M. Sablé-Meyer, L. Morales, L. B. Hewitt, L. Cary, A. Solar-Lezama, J. B. Tenenbaum, Dreamcoder: bootstrapping inductive program synthesis with wake-sleep library learning, Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation (2021).

[3] S. Gulwani, Automating string processing in spreadsheets using input-output examples, in: POPL '11, 2011.

[4] L. Jiao, J. Gao, X. Liu, F. Liu, S. Yang, B. Hou, Multi-scale representation learning for image classification: A survey, IEEE Transactions on Artificial Intelligence (2021) 1–1. doi:10.1109/TAI.2021.3135248.

[5] Y. Yu, X. Si, C. Hu, J. Zhang, A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures, Neural Computation 31 (2019) 1235–1270. URL: https://doi.org/10.1162/neco_a_01199. doi:10.1162/neco_a_01199. arXiv:https://direct.mit.edu/neco/article-pdf/31/7/1235/1053200/neco_a_01199.pdf.

[6] F. Chen, Y.-C. Wang, B. Wang, C.-C. J. Kuo, Graph representation learning: A survey, APSIPA Transactions on Signal and Information Processing 9 (2020).

[7] P. Veličković, C. Blundell, Neural algorithmic reasoning, Patterns 2 (2021) 100273. doi:https://doi.org/10.1016/j.patter.2021.100273.

[8] D. R. Hofstadter, Fluid concepts and creative analogies: Computer models of the fundamental mechanisms of thought., Basic books, 1995.

[9] F. Chollet, On the measure of intelligence, ArXiv abs/1911.01547 (2019).

[10] D. Barrett, F. Hill, A. Santoro, A. Morcos, T. Lillicrap, Measuring abstract reasoning in neural networks, in: J. Dy, A. Krause (Eds.), Proceedings of the 35th International Conference on Machine Learning, volume 80 of *Proceedings of Machine Learning Research*, PMLR, 2018, pp. 511–520. URL: https://proceedings.mlr.press/v80/barrett18a.html.

[11] B. M. Lake, R. Salakhutdinov, J. B. Tenenbaum, Human-level concept learning through probabilistic program induction, Science 350 (2015) 1332 – 1338.

[12] B. M. Lake, M. Baroni, Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks, in: ICML, 2018.

[13] N. Rahaman, M. W. Gondal, S. Joshi, P. V. Gehler, Y. Bengio, F. Locatello, B. Scholkopf, Dynamic inference with neural interpreters, ArXiv abs/2110.06399 (2021).

[14] V. Kolev, B. Georgiev, S. Penkov, Neural abstract reasoner, ArXiv abs/2011.09860 (2020).

[15] J. Johnson, B. Hariharan, L. van der Maaten, J. Hoffman, L. Fei-Fei, C. L. Zitnick, R. B. Girshick, Inferring and executing programs for visual reasoning, 2017 IEEE International Conference on Computer Vision (ICCV) (2017) 3008–3017.

[16] P. Velickovic, R. Ying, M. Padovano, R. Hadsell, C. Blundell, Neural execution of graph algorithms, ArXiv abs/1910.10593 (2020).

[17] L.-P. Xhonneux, A. Deac, P. Velickovic, J. Tang, How to transfer algorithmic reasoning knowledge to learn new algorithms?, ArXiv abs/2110.14056 (2021).

[18] S. van Steenkiste, F. Locatello, J. Schmidhuber, O. Bachem, Are disentangled representations

helpful for abstract visual reasoning?, in: NeurIPS, 2019.
[19] X. Steenbrugge, S. Leroux, T. Verbelen, B. Dhoedt, Improving generalization for abstract reasoning tasks using disentangled feature representations, ArXiv abs/1811.04784 (2018).
[20] F. Hill, A. Santoro, D. Barrett, A. Morcos, T. Lillicrap, Learning to make analogies by contrasting abstract relational structure, in: International Conference on Learning Representations, 2019. URL: https://openreview.net/forum?id=SylLYsCcFm.
[21] R. Silva, K. A. Heller, Z. Ghahramani, Analogical reasoning with relational bayesian sets, in: AISTATS, 2007.
[22] D. Gentner, Structure-mapping: A theoretical framework for analogy*, Cognitive Science 7 (1983) 155–170.
[23] M. Crouse, C. Nakos, I. Abdelaziz, K. D. Forbus, Neural analogical matching, ArXiv abs/2004.03573 (2021).
[24] A. Sonwane, G. Shroff, L. Vig, A. Srinivasan, T. Dash, Solving visual analogies using neural algorithmic reasoning, CoRR abs/2111.10361 (2021). URL: https://arxiv.org/abs/2111.10361. arXiv:2111.10361.
[25] M. Garnelo, D. Rosenbaum, C. Maddison, T. Ramalho, D. Saxton, M. Shanahan, Y. W. Teh, D. Rezende, S. M. A. Eslami, Conditional neural processes, in: J. Dy, A. Krause (Eds.), Proceedings of the 35th International Conference on Machine Learning, volume 80 of *Proceedings of Machine Learning Research*, PMLR, 2018, pp. 1704–1713. URL: https://proceedings.mlr.press/v80/garnelo18a.html.
[26] O. Vinyals, C. Blundell, T. Lillicrap, k. kavukcuoglu, D. Wierstra, Matching networks for one shot learning, in: D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, R. Garnett (Eds.), Advances in Neural Information Processing Systems, volume 29, Curran Associates, Inc., 2016. URL: https://proceedings.neurips.cc/paper/2016/file/90e1357833654983612fb05e3ec9148c-Paper.pdf.
[27] C. Finn, P. Abbeel, S. Levine, Model-agnostic meta-learning for fast adaptation of deep networks, in: Proceedings of the 34th International Conference on Machine Learning - Volume 70, ICML'17, JMLR.org, 2017, p. 1126–1135.

# A. Details of Methodology

## A.1. Problem Setting

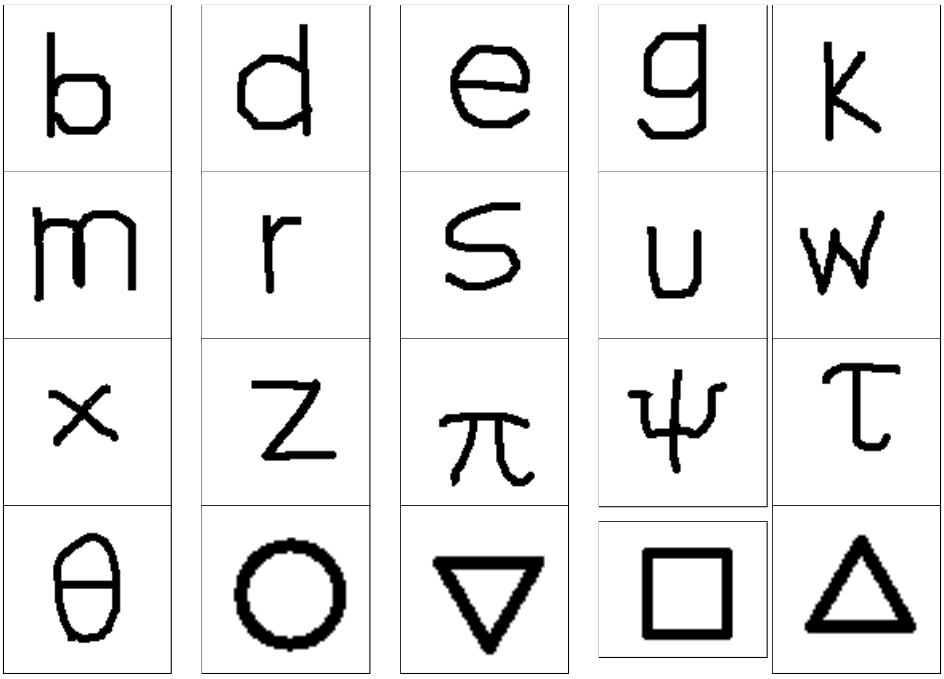


**Figure 6:** The set of base shapes and characters (modulo rotations and skews) used to construct visual analogy tasks.

| Types | Transformations | Meaning |
|---|---|---|
| Shifts | `shift-right`, `shift-left`, `shift-up`, `shift-down` | Move all of the shapes in the image by one grid positions |
| Shape conversions | `to-square`, `to-circle`, `to-delta`, `to-triangle` | Convert all of the shapes in the image to a particular shape |
| Affines | `affine-right`, `affine-left` | Skew all the shapes in the image either to the left or right |

**Table 3**
All spatial transformation used in visual analogy tasks

**Details of Images used in Visual Analogy Tasks** Each image used in the visual analogy task consists of shapes or characters from the set described in Figure 6 placed on a $3 \times 3$ grid. The size of each such image is $64 \times 64$ pixels with the the size of shapes and characters placed inside being $20 \times 20$ pixels. Note that the grid isn't explicitly drawn and all images are black and white (the background being white and the shapes being solid black). The handwritten characters used have been taken from the Omniglot dataset.

## A.2. Guided Representation Space

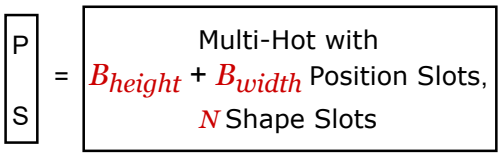



**Figure 7:** Symbolic descriptions of input images are in the form of multi-hot vectors constructed by concatenating a one-hot vector denoting which shape is present in the image and another one-hot vector denoting the position of the shape in the image. With 20 possible shapes and a $3 \times 3$ grid, this multi-hot vector will be of length 26.

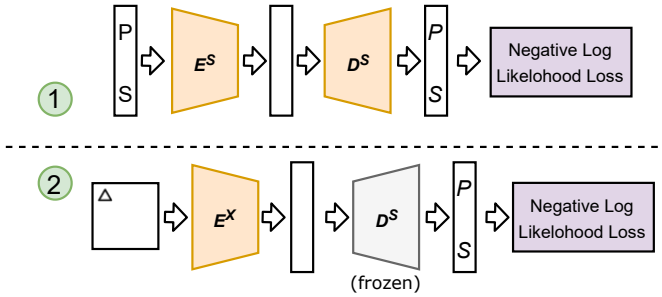



**Figure 8:** Two step procedure for obtaining image encoder using latent representation space based on symbolic descriptions

**Representation Learning** For the *guided* representation space we first train an autoencoder directly on the symbolic descriptions (as described in Figure 7) of images containing a single shape followed by training an encoder from said images to the latent embeddings of their symbolic descriptions. This procedure is described in Figure 8 The size of the latent embeddings here is 32. The encoder and decoder used for the autoencoder over symbolic descriptions are both MLPs with a single hidden layer of size 16. They are trained using batch size of 36 for 10,000 weight updates with a learning rate of 0.0003. For the encoder from images to the latent embeddings of their symbolic descriptions we use a CNN with 6 convolutional layers each using a stride of 2, padding of 1 and $3 \times 3$ kernels. Each layer doubles the number of channels and halves the dimensions of its input and applies a rectified linear unit activation. Since the input image has dimensions $1 \times 64 \times 64$, the output of these six layers is $128 \times 1 \times 1$. This is flattened and passed through a final linear transformation to output the latent embedding of size 32. This encoder is trained with a batch size of 180 (since there are $20 \times 9 = 180$ possible combinations of shape and grid positions) for 50,000 weight updates with a learning rate of 0.0003.

**Neural Transforms** For the *guided* representation space, we use MLPs as neural transforms with a single hidden layer of size 32. These are trained with a batch size of 180 for 5000 weight updates with a learning rate of 0.0003. A overview of the training procedure is given in Figure 9.

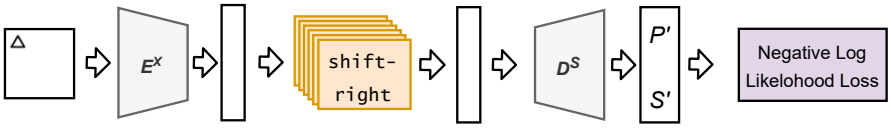



**Figure 9:** Procedure for training transforms for *guided* representation space

## A.3. Unguided Representation Space

**Representation Learning** For the *unguided* representation space we train an autoencoder directly on the images containing a single shape with latent embedding size of $16 \times 8 \times 8$. The encoder used is a CNN with 3 convolutional layers each followed by a batch normalisation layer. Each layer uses a stride of 2 and padding of 1. The first two layers use $4 \times 4$ kernels and the last one uses $3 \times 3$ kernel. The first layer outputs 8 channels while the second and third both output 16 channels. The decoder is made up of 3 transpose convolution layers each followed by batch normalisation. All three layers use a stride of 2, padding of 1 and $4 \times 4$ kernels. The first two layers output 8 channels. The autoencoder is trained using batch size of 32 for 20000 weight updates with a learning rate of 0.001.

**Neural Transforms** For the *unguided* representation space we use MLPs as neural transforms with a single hidden layer of size 256 for shift and shape conversion transforms and of 2048 for affine transforms (although similar performance is observed for size 256). These are trained with a batch size of 18 for 4000 weight updates with a learning rate of 0.001 on dataset of images with a single shape present. Note that the latent embeddings are flattened to a 1024 long vector before being passed to the transforms.

## A.4. Program Execution Algorithm

**Algorithm 1** Program Execution

**Input**: Input latent vectors $z_k$s, program $P$
**Output**: Output latent vectors

input = $\{z_k\}$, memory = $\{z_k\}$, output = $\emptyset$
**for** $i = 1 : \text{length}(P)$ **do**
  **if** $P[i]$ = `out` **then**
    output $\leftarrow$ output $\cup$ memory
  **else if** $P[i]$ = `reset` **then**
    memory $\leftarrow$ input
  **else**
    memory $\leftarrow \{P[i](z_k), \forall z_k \in \text{memory}\}$
  **end if**
**end for**
**return** output

## A.5. Search Algorithm

**Algorithm 2** Limited Depth BFS

**Input** Encoder: $E$, Decoder: $D$ Transforms: $T = \{T_i\}$, Example: $\lambda = (x_i, x_o)$
**Output** Program satisfying all given examples

```
targets ← connected_components(x_o)
q ← LIFO();  q.push([ ])
max_count ← 0
while True do
    program ← q.pop();
    if length(program) = d then
        return None;
    end if
    for t_i ∈ T^Z do
        new_program ← program + t_i;
        if satisfies(λ, new_program, E, D) then
            return new_program;
        end if
    end for
end while
```

**Caching program state to speed up search** Alongside pruning we also reduce the constant factor involved at each step of the search by keeping track of the program state after partial execution. This creates a significant reduction in time taken since the majority of it is the neural network forward passes. This means that transforms are only applied when a new transform is added to the program. We do the same for output decoding - only applying output decoder whenever the `out` primitive is called and keeping track of output buffer in subsequent nodes.

## A.6. Example Run-through

Consider the example task in the shown in Figure 10. An input image $x_i$ and target image $x_r$ makes up the example for an unknown transformation sequence which needs to be applied to the query image $x_q$. We have an encoder $E_x$ and decoder $D_x$ trained as described in section 3 and a set of transforms $T_i$ trained according to section 3.

Solving this task using the *guided* approach, the first step is isolating connected components in the input image. This results in two images for the input in our example $x_i^{(1)}$, $x_i^{(2)}$, each of which is converted into its latent representation using the encoder and stored in the input and memory buffers: $I = M = \{z_i^{(1)}, z_i^{(2)}\}$. Having obtained the initial program state, the search procedure enumerates sequences of transforms which it tests against the example.

Suppose that the search arrives at `shift-left reset shift-down out`. We first apply the neural network $T_{\texttt{shift-left}}$ to the latent vectors present in the memory buffer $z_i^{'(1)} = T_{\texttt{shift-left}}(z_i^{(1)})$ and $z_i^{'(2)} = T_{\texttt{shift-left}}(z_i^{(2)})$. Here $z_i^{'(1)}$ and $z_i^{'(2)}$ are now latent embeddings of the characters in

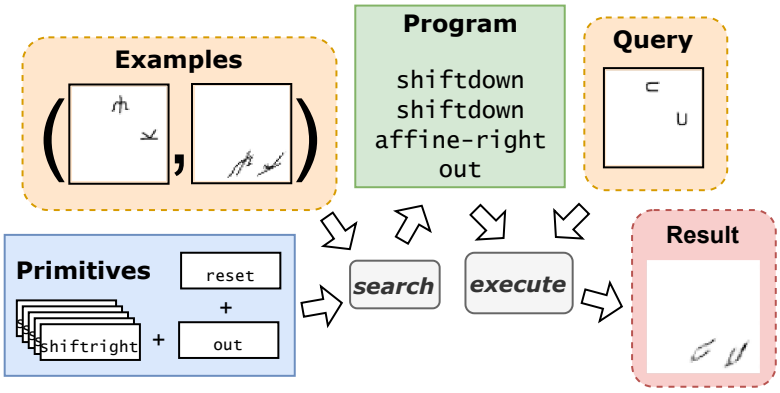



**Figure 10:** Demonstration on an example task.

the original image but shifted to the left by one grid position. `reset` is a special primitive that results in the memory buffer being reset with the contents of the input buffer: $M = I = \{z_i^{(1)}, z_i^{(2)}\}$. The application of `shift-down` proceeds similarly to that of `shift-left` using the appropriate neural network transform. Finally `out` is another special primitive which adds all the latent vectors present in the memory buffer to the the output buffer: $O = \{z_i''^{(1)}, z_i''^{(2)}\}$. After executing the program, we use decoder $D_x$ to produce the images corresponding to the latent vectors present in the output buffer which are then added together to form the output image. Since this resulting image would contain the same characters as in the input image but shifted down by one grid position, comparison with the target using a distance function as described in section 3 would lead us to reject the current transform sequence and keep searching. Eventually, the search arrives at a sequence that satisfies the example, which is our case is `shift-down shift-down affine-right out`. This is then applied to the query image to get our final result.

## A.7. Data Generation

Random example (input, output) pairs can be generated by applying a randomly generated program to a randomly selected image. During generation of the program, constraints are applied to minimise the generation of any redundant programs. These are (1) no consecutive shape conversion transforms, (2) no consecutive `out` or `reset` operations, (3) all programs must end with an `out` operation and (4) `reset` operations must follow an `out` operation. When generating a set of random examples, it is also ensured that all examples generated are unique.

## A.8. Details of Evaluation Dataset

To evaluate the ability of transforms in our system to compose without error, we use of a dataset of (input, output) pairs (generated from a dataset of 20000 programs, 200 programs each for lengths of 1 till 10) and compute the ratio of pairs for which a valid program is found. These pairs are generated from a dataset of 20000 programs, 200 programs each for lengths of 1 till 10. Each task is one-shot (contains a single input-output pair), and the images can contain multiple shapes. The programs are made up of 4 types of `shift` transforms and 2 `affine` transforms. We

exclude shape conversions from our experiments since their application to an unseen shape turns it into a seen shape, thus not allowing us to properly test for generalization performance. Also, programs containing `affine` transforms were not used in evaluation for the *guided* case since, as discussed in Section 3, continuous shape transformations apply only in the *unguided* case.

## B. Details of Meta Learning Implementations

We cast our analogical reasoning task into a few shot classification problem - each task consists of 6 examples representing a single transformation, where each example consists of an input image, 4 options for the possible outputs and an index indicating the correct output as shown in Figure 11.

The task also contains a query image and 4 option images for that query. This is further deconstructed into a binary classification problem - for each example, we compute the pixel difference between the input image and each of the 4 option images. Each such pixel difference is assigned a binary label resulting in a $6 \times 4 = 24$ meta-train examples and 4 meta-test queries for each task.

We used custom made Convolutional Neural Networks to get the features of the input images. CNP and MAN were run for 60 epochs whereas MAML is run for 30 epochs. MAML uses a dropout of 0.5 and CNP uses a dropout 0.8. We used ReduceLROnPlateau learning rate scheduler with a threshold of $10^{-3}$ modifying the other parameters as required. Both Dropout and Learning rate scheduler were essential to reduce the fluctuations in the training loss.

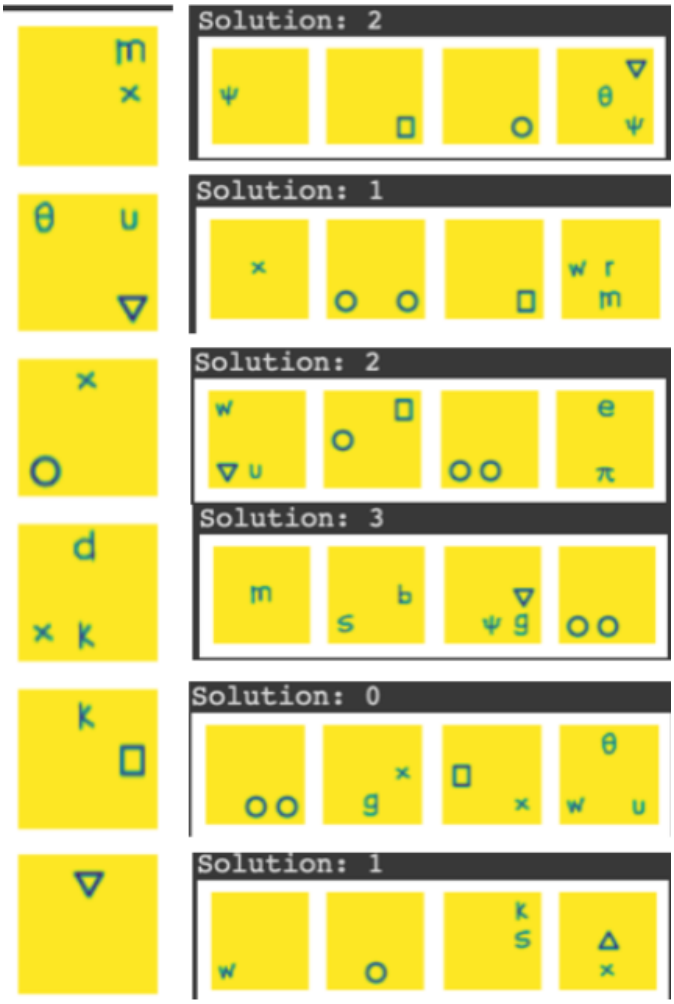



**Figure 11:** Visualization of each task of the few-shot classification dataset use to evaluate end-to-end meta learners. Each row is an example, containing an input image, 4 possible output images and index (starting from 0) of the correct choice. The transformation here is `shift-down, shift-down, to-circle` which corresponds to shifting down the objects present in the input image by two grid positions and replacing them with circles.

# C. Further Results

## C.1. Evaluations of the representation space

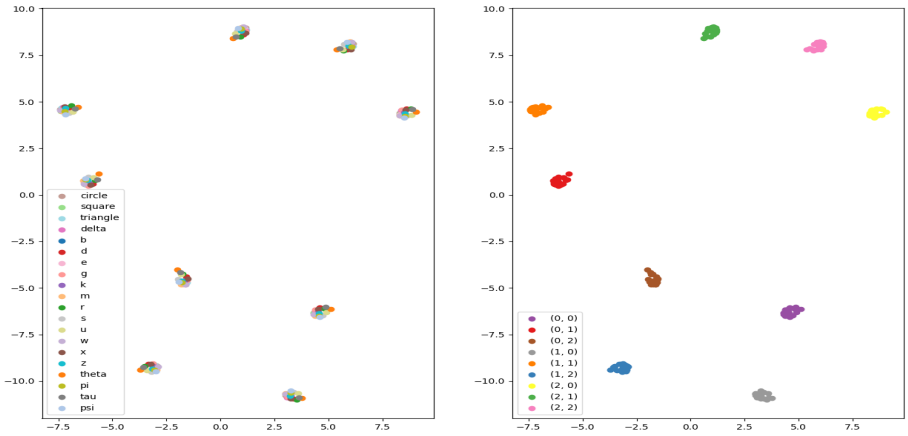



**Figure 12:** t-SNE Plots for latent embedding of autoencoder in the *guided* case trained with mean-squared distance loss as opposed to the log-likelihood loss which is used in the final system. Each point represents an embedding for a specific shape in a specific grid position. The left hand plots have points coloured according to shape and right hand plots according to grid position. We can see that the clusters for each position are much more tightly grouped than when using log-likelihood loss making it much harder to distinguish between shapes in the same position.

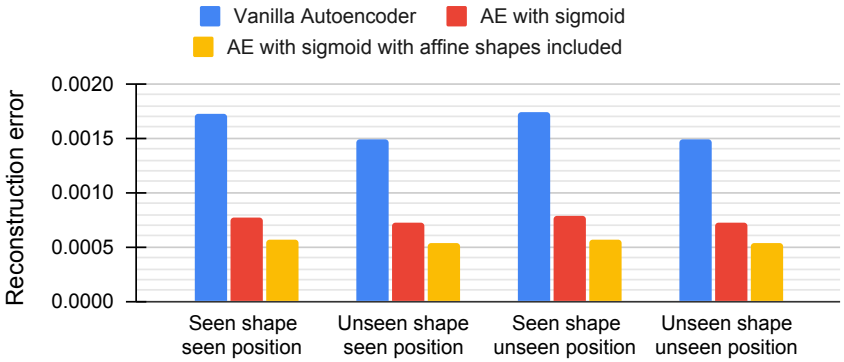



**Figure 13:** Reconstruction error for different types of autoencoders evaluated on different data regimes in the *unguided* case. The prefix *seen* indicates that the error being reported is over variants of the particular attribute (shape or position) that were present during training and *unseen* indicates that the error is over those variants that were not present during training. For example the *unseen shape seen position* case indicates the reconstruction error for shapes that were not present during training of the autoencoder but in positions that were included during training. Here *with sigmoid* denotes the autoencoder uses the sigmoid activation on the output layer and *with affine* denotes that images with affine transformations of the shapes and characters were included during training of the autoencoder. These results indicate that reconstruction is improved drastically by the use of sigmoid and also somewhat by more diverse training data.

## C.2. Ability of Neural Transforms to Compose

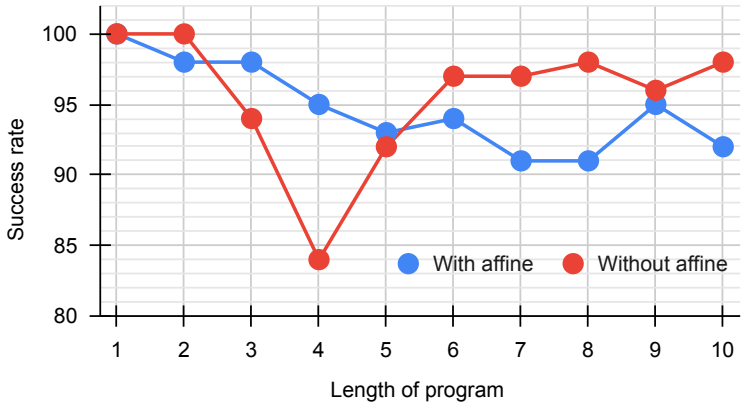



**Figure 14:** Average success ratio of system using *unguided* representation space on varying program lengths. The *with* and *without affine* indicates whether the dataset of programs being evaluated on contained affine transforms.

## C.3. Evaluation of *Guided* Representation Space Generalisation

To evaluate how well the *guided* representation space (encoder and decoder) generalises to unseen situations, we utilise the following scheme. Let us a tuple $(M, N, R, T) \in \mathscr{P}(Q)^4$ where $Q$ is the set of all shapes as described in Figure 6, $M$ denotes the shapes present in the training data for the latent space (here $|M| + 1 + 6$ denotes the size of the multi-hot vector, where 6 is for the positional encoding and 1 is for a slot of *unseen* shape), $N$ denotes the shapes present in the training data for transforms, $R$ denotes the shapes present in the training data for the encoder and $T$ denotes the shapes present in the examples on which the latent space is being evaluated. We perform the following 4 experiments based on this evaluation scheme to test the generalisation capabilities of the *guided* representation space. The results are given in Figure 15. Note that the total number of shapes possible $|Q| = 20$.

1. For $M = N = \emptyset$ (all shapes considered unseen in the multi-hot) and $T = Q$ (all shapes included in test examples), we vary the shapes shown to the CNN encoder during training from $|R| = 1$ (just one shape being included in training data) $R = Q$ (all shapes included) and record the evaluation performance.

2. For $M = N = \emptyset$ (all shapes considered unseen in the multi-hot), we vary the shapes shown to the CNN encoder during training from $|R| = 1$ (just one shape being included in training data) $R = Q$ (all shapes included) and record the evaluation performance of with this encoder on $T = R'$ (test only including shapes not present in train for encoder).

3. For $M = N = \{\text{square}, \text{triangle}, \text{circle}, \text{delta}\}$ (four shapes + unseen in the multi-hot) and $T = Q$ (all shapes included in test examples), we vary the shapes shown to the CNN

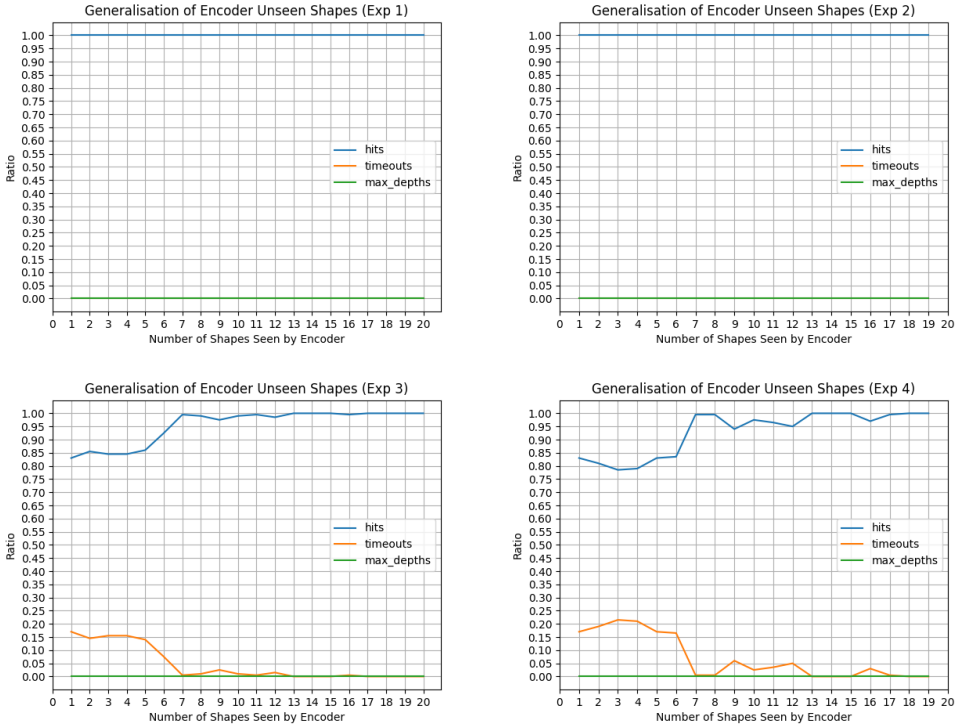



**Figure 15:** Generalisation of *guided* representation space to unseen shapes

encoder during training from $|R| = 1$ (just one shape being included in training data) $R = Q$ (all shapes included) and record the evaluation performance.

4. For $M = N = \{\text{square}, \text{triangle}, \text{circle}, \text{delta}\}$ (four shapes + unseen in the multi-hot), we vary the shapes shown to the CNN encoder during training from $|R| = 1$ (just one shape being included in training data) $R = Q$ (all shapes included) and record the evaluation performance of with this encoder on $T = R'$ (test only including shapes not present in train for encoder).

## C.4. Evaluation of Transform Generalisation in *Guided* Representation Space

To evaluate how well neural transforms learned in the *guided* representation space generalise to unseen situations, we first learn the representation space with data containing all of the shapes or positions. We then train the neural transforms on (input, output) image pairs containing only a subset of shapes (or input positions). During testing, we include all shapes (or positions). This can be thought of as keeping $M = R = T = Q$ while $N$ is varied from $Q$ to $\emptyset$. If the transforms give similar performance to those trained with all the data, we can say that they, together with the latent space, are generalising well. The results are given in Figure 16.

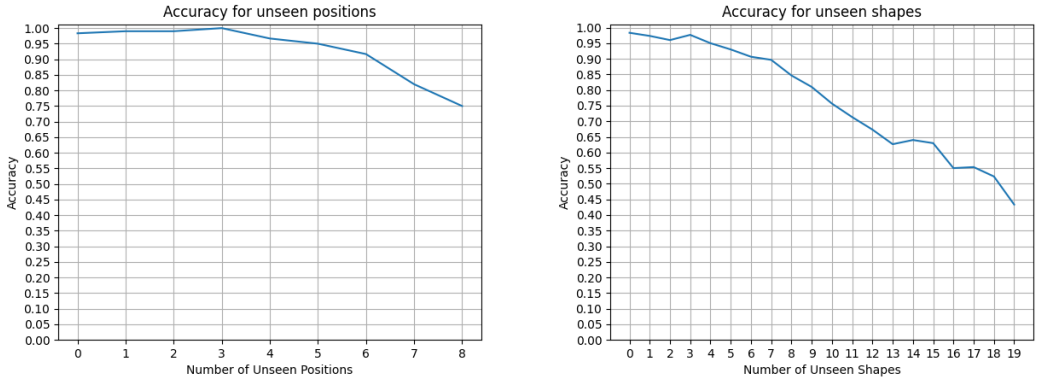



**Figure 16:** Generalisation of transforms to unseen attributes for *guided* representation space

## C.5. Evaluation of *Unguided* Representation Space Generalisation

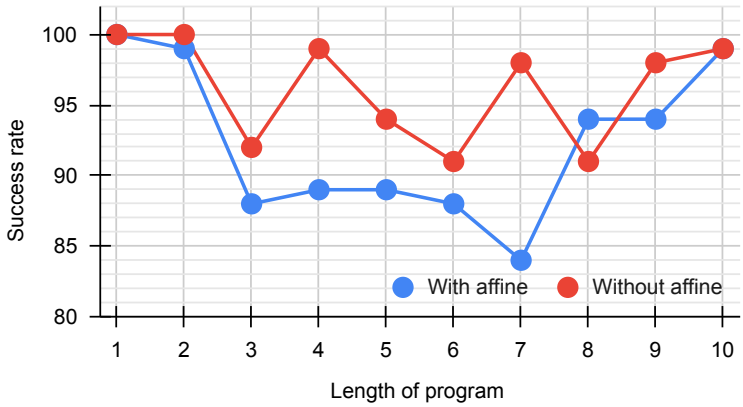



**Figure 17:** Average success ratio of system using *unguided* representation space on varying program lengths on dataset of (input, output) examples containing shapes that were not seen during training. Out of the 20 total shapes, 15 were present during training and all 20 were present in the evaluation dataset. The *with* and *without affine* indicates whether the dataset of programs being evaluated on contained affine transforms.

## C.6. Evaluation of Search Performance

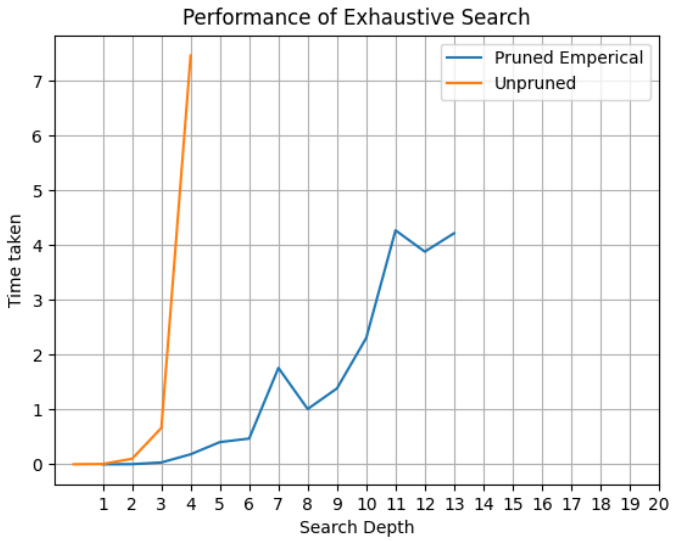



**Figure 18:** Comparison between the running time of naive and pruned BFS for program search. We can see that pruning leads to significant performance gains enabling the system to search for longer programs within shorter time budgets. While this plot is for the *guided* representation case, similar behaviour is observed for the *unguided* case since both use the same search procedure and pruning measures.