Analogical and Relational Reasoning with Spiking Neural Networks

Rollin Omari, R. I. (Bob) McKay and Tom Gedeon

***Abstract*—Raven's Progressive Matrices have been widely used for measuring abstract reasoning and intelligence in humans. However for artificial learning systems, abstract reasoning remains a challenging problem. In this paper we investigate how neural networks augmented with biologically inspired spiking modules gain a significant advantage in solving this problem. To illustrate this, we first investigate the performance of our networks with supervised learning, then with unsupervised learning. Experiments on the RAVEN dataset show that the overall accuracy of our supervised networks surpass human-level performance, while our unsupervised networks significantly outperform existing unsupervised methods. Finally, our results from both supervised and unsupervised learning illustrate that, unlike their non-augmented counterparts, networks with spiking modules are able to extract and encode temporal features without any explicit instruction, do not heavily rely on training data, and generalise more readily to new problems. In summary, the results reported here indicate that artificial neural networks with spiking modules are well suited to solving abstract reasoning.**

***Index Terms*—Raven's progressive matrices (RPM), spiking neural network (SNN), liquid state machine (LSM), unsupervised learning.**

I. Introduction

In the field of Artificial Intelligence (AI), one of the most important milestones is the construction of machines with human-level reasoning abilities. To achieve this goal, various datasets and tasks have been introduced into the field to enable training and evaluation of AI systems within this scope. One particular task that has become increasingly popular is the training and evaluation of AI systems on Raven's Progressive Matrices (RPM). RPMs are one of the most popular instruments for measuring abstract reasoning and fluid intelligence in humans as they require subjects to solve problems in the absence of physical objects, or concrete phenomena and independent of their language, reading and writing skills, or even their cultural background [1], [2].

As shown in Figure 1, an RPM consists of several visual geometric designs with a missing piece. Given eight candidate choices as the answer set, one has to determine the underlying logical rules in the problem matrix and select the best choice from the answer set that satisfies these hidden rules [3]. Within the context of machine intelligence and in contrast to other computer vision tasks [4], RPMs require that computers generalise about relations and attributes between abstract objects. More importantly, RPMs require that computers make sense of unforeseen patterns with limited amounts of data, and broadly generalise acquired information to many tasks.

To help make progress towards solving RPMs with artificial intelligence, we focus on simulating and evaluating relational and analogical reasoning in Artificial Neural Networks (ANNs). Specifically, unlike existing methods that evaluate machine intelligence on RPMs [5], [6], [7] or works that focus on object representation and form recognition [8], [9], [10], we decouple visual processing from our networks and aim to directly emulate the reasoning capabilities of the human prefrontal cortex [11]. However, beyond simply emulating the human prefrontal cortex, we focus on two biologically inspired neural networks, each using a different learning strategy.

In our first experiment, we focus on supervised learning with a computationally light neurally inspired algorithm known as a reservoir computer, implemented in the form of a Liquid State Machine (LSM). An LSM is a neural network introduced by Maass *et al.* [12] that consists of an input layer, a liquid layer, and a readout layer. One advantage of an LSM is that the synaptic connections in the liquid layer have an inherent time-dependency, since any information the layer encodes fades out over time [13]. A second, arising from their time-dependency, is the ability of the recurrent connections in the liquid layer to capture high-dimensional dynamic information. A third advantage of an LSM is that all the synaptic connections, other than those connecting to the readout layer, are randomly initialised and remain fixed. Hence unique inputs will produce distinct perturbations in the state of the high-dimensional liquid layer, from which information can be extracted by the readout layer [13].

In the context of solving RPMs with neural networks, we demonstrate that these advantages make LSMs great candidates, as RPMs inherently consist of unique high-dimensional dynamic information. In particular, while using features extracted by the Dynamic Residual Tree (DRT) module [14], our LSM model attains a testing accuracy of 93.30%, which surpasses human-level performance (84.41%) and the current sate-of-the-art performance (CoPINet, 91.42%) on the RAVEN dataset [14].

In our second experiment, we focus on unsupervised learning with two Spiking Neural Networks (SNNs), each employing leaky integrate-and-fire (LIF) neurons with adaptive thresholds. Both networks consist of an input layer and an output layer, with recurrent connections in the output layer. For the first SNN model, the forward connections between

R. Omari is with the Defence Science and Technology Group, Australian Department of Defence, Edinburgh, SA, 5111 Australia, and also with the Research School of Computer Science, CECS, The Australian National University, Canberra, ACT, 2601 Australia.

R. McKay and T. Gedeon are with the Research School of Computer Science, CECS, The Australian National University, Canberra, ACT, 2601 Australia. (e-mail: {rollin.omari, robert.mckay, tom.gedeon}@anu.edu.au).

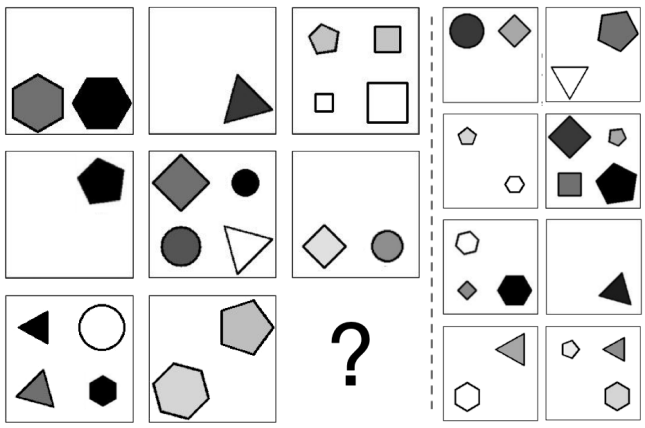

Fig. 1: An example of an RPM problem in RAVEN [14]. To complete the problem matrix, one has to select the best choice in the answer set that follows structural and analogical relations. In this problem, the *positions*, *sizes*, *colours* and *shapes* can vary freely as long as the *number* of the shapes follows the underlying rule.

the input and output layer utilise the standard Spike-Timing Dependent Plasticity (STDP) learning algorithm, similarly for the recurrent connections in the output layer. For the second SNN model the forward connections between the input and output layer are similar to the first SNN model. However, the recurrent connections in the output layer utilise a learning algorithm introduced by Hazan *et al.* [15], which combines the local learning rules from STDP and the clustering properties from a Self-Organizing Map (SOM) algorithm.

Beyond simply making contributions towards solving RPMs with ANNs, our experiments with spiking networks illustrate how spikes potentially play an essential role in information processing and data representation in cognitive systems. In fact, our spiking models attain testing accuracies of 54.15% (SNN) and 71.04% (SSOM) without using features extracted from the DRT module. When using features from the DRT module, the models attain respective testing accuracies of 84.86% (SNN) and 93.23% (SSOM), drastically outperforming the unsupervised MCPT method (28.50%). These results, when taken into consideration with our LSM results, also indicate how network architecture plays an essential role in solving RPMs with ANNs and attaining human-level performance.

The remainder of this paper is organised as follows. In the next section, we discuss related works in analogical reasoning and computational efforts in generating and solving RPMs, while Section III provides a formal description of solving RPMs. In Section IV, we offer a detailed explanation of our Liquid State Machine and Spiking Neural Network models. In Section VI, we offer an explanation of our experimental configurations, and report and analyse our results on the RAVEN dataset. We finally conclude this paper with section VII and offer remarks about future research.

II. Related Work

In recent years, Raven's Progressive Matrices have become widely used to test the capability of abstract reasoning in artificial neural networks. Inspired by John Raven, Santoro *et al.* [5] released the first large-scale RPM dataset named Procedurally Generated Matrices (PGM), and introduced the Wild Relation Network (WReN) as a potential method for solving RPMs in this dataset. WReN was designed to formulate pairwise relations between the problem matrix and each individual choice in an embedding space, however earlier versions of WReN had limited generalization performance. Steenbrugge *et al.* addressed this problem by incorporating a pre-trained Variational Auto Encoder [16].

Recently, Zhang *et al.* [14] generated a new RPM dataset named RAVEN and also introduced a Dynamic Residual Tree (DRT) module that considers annotations of image structure and thereby symbolizing a problem matrix. Following their introduction of DRT, Zhang *et al.* also introduced a new method known as CoPINet, which combines contrasting, perceptual inference and permutation invariance [6]. More recently, Wang *et al.* introduced a multiplex graph to capture multiple relations between objects [17], while Zhuo and Kankanhalli [7] modified a ResNet-50 network with ImageNet pre-training to reduce over-fitting, and they also proposed MCPT as a method for solving RPM problems in an unsupervised manner.

III. Problem Formulation

As mentioned in the introduction, an RPM is a task designed to measure non-verbal, cognitive, and abstract reasoning. Formally, this task involves a problem matrix $\mathbf{x}$ consisting of images $\{x_{11}, x_{12}, \ldots, x_{33}\}$ and a corresponding answer set $\mathbf{y}$. Solving an RPM problem involves selecting the best answer $\mathbf{y}_a$ to complete the question matrix $\mathbf{x}$, where a is the index of $\mathbf{y}_a$ and $a \in \{1, 2, \ldots, 8\}$.

Given T training samples $\{(\mathbf{x}_i, \mathbf{y}_i, a_i)\}_{i=1}^{T}$ with $\mathbf{x}_i \in X$, $\mathbf{y}_i \in Y$ and $a_i \in A$, we aim to learn a function f over (X, Y) and A. In our implementation, all images in the problem matrix and its corresponding answer set are stacked together, and then fed into a neural network for the final answer prediction. The learning method can be formulated as:

$$A = f(\phi(X \cup Y); \mathbf{w}), \tag{1}$$

where $\cup$ denotes the concatenation operation that stacks all images in the question matrix and the answer set; ϕ represents the image features over $(X \cup Y)$; $\mathbf{w}$ is the parameter set to learn. Given this formulation, an RPM problem can be treated as a multi-class classification task.

IV. Approach

In this section we introduce our neural networks and their respective learning approaches.

A. Feature Extractor

For our feature extractor, we use a ResNet-18 model with a multi-layer perceptron (MLP) module. The MLP module has an input layer configuration of 512×100 and an output layer configuration of 100×8. We use an Adam optimiser to learn

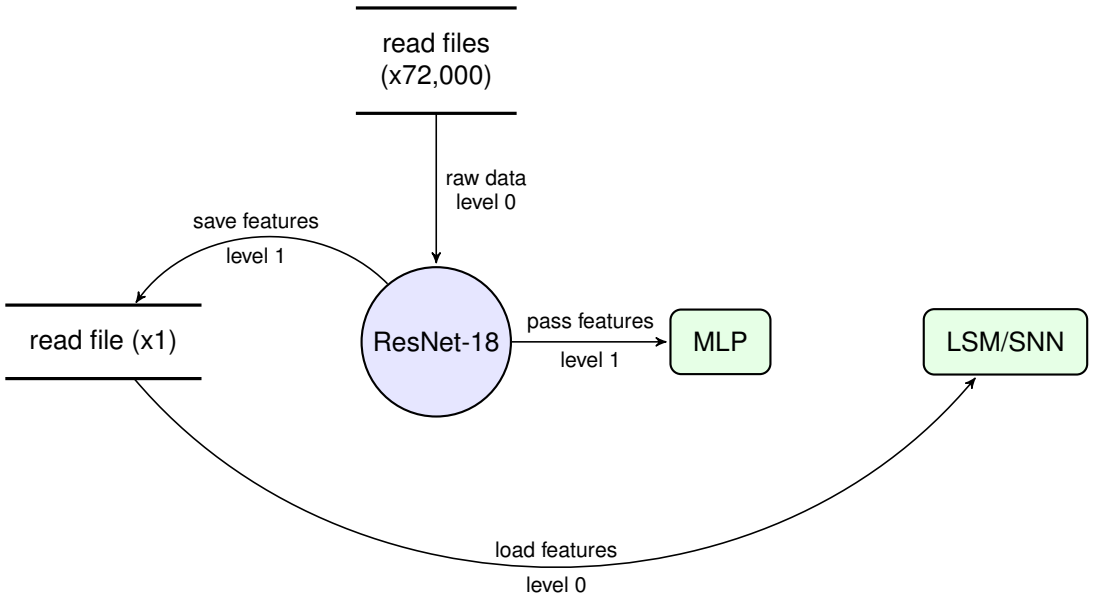

Fig. 2: Schematic for dataset processing and feature extraction. RPM images are loaded into a ResNet-18 network and processed. The corresponding features for each RPM is extracted from the final layer of the ResNet-18 network, placed into a large array and saved into a single file.

network parameters [18]. And we compute the cross-entropy loss for each RPM with

$$CE(\mathbf{w}) = -\sum_{c=1}^{n} a_c \log(p_c), \tag{2}$$

where $\mathbf{w}$ is the parameter set to learn, $p_c = f_c(\phi(\mathbf{x}_i \cup \mathbf{y}_i); \mathbf{w})$ is the model's estimated probability for the class with label c. The estimated probability p_c is obtained by the model by using the learned function f_c, image features ϕ over the stack $\cup$ of the question matrix $\mathbf{x}_i$ and its corresponding answer $\mathbf{y}_i$.

For the majority of our experiments, we use this feature extractor as illustrated in Figure 2. However, for a few experiments we place a DRT module between the ResNet-18 module and the MLP module, and use the extracted features from the DRT module. For experiments in which DRT features are utilised, we designate those models as "Model+DRT", and leave their designation as normal otherwise. By using extracted features rather than raw images, we decrease the running times for each of our models by about $200\times$, but more importantly, we enable direct comparison of analogical reasoning in models, without any interference from feature learning.

B. Liquid State Machine

For our LSM, we use a three-layered network that consists of an input layer, liquid layer and output layer. The input layer is an array of $M \times N$ neurons, which corresponds to the number of pixels of the input images, or more generally, the number of features in the input. Each neuron in the input layer is a spiking neuron that sets its spike occurrences v equal to its respective inputs $I(t)$:

$$\tau_m \frac{dv}{dt} = I(t), \tag{3}$$

where τ_m is the membrane time constant. In our case, the spikes in the input layer are generated using a Bernoulli spiking process, with average firing rate τ proportional to pixel intensity [15]. This process continues for a specific duration and the STDP rule is executed for connections between the input layer and the liquid layer.

For the liquid layer, our LSM consists of K neurons whose dynamics are modelled by a simple leaky integrate-and-fire model [19]. Specifically, in the liquid layer, a neuron is described as a simple resistor-capacitor circuit:

$$\tau_m \frac{dv}{dt} = -v(t) + RI(t), \tag{4}$$

where $v(t)$ is the membrane potential at time t, and R is the membrane resistance. To allow for actual spiking events, this model is augmented with a spiking threshold v_θ, where if $v(t)$ reaches this value, it is instantaneously reset to the resting potential v_r. To add some realism to this model, an absolute refractory period is included, where immediately after $v(t)$ reaches v_θ, $v(t)$ is clamped to v_r and the leaky integration process starts again after a delay of a few milliseconds. For connections in the liquid layer, we make these recurrent and use the STDP learning rule for weight update.

Finally, for the output layer we use a simple sigmoid linear regression layer that consists of $K \times L$ neurons, whose inputs are spikes collected from the liquid layer. For the connections in the output layer, we use Stochastic Gradient Descent (SGD) and the mean squared error (MSE) loss to learn parameters.

C. Spiking Neural Network

For our SNN, we use a two-layered network that consists of an input layer and an output layer. For the input layer, we use the same process as described in Section IV-B with no additional changes. As for the output layer, we use a leaky integrate-and-fire model with adaptive thresholds. The dynamics for the neurons in the output layer are largely described by Equation 4, however for details on the adaptive thresholds and the remaining parameters used for this particular LIF model, we refer the interested reader to [20].

The connections between the input layer and output layer are as described in the previous section. As for the recurrent connections in the output layer, we use the architecture and unsupervised learning algorithm developed by [15]. Specifically, the recurrent connections in the output layer combine STDP and a self-organizing map algorithm. This unsupervised learning algorithm encourages the output layer to self-organise into distinct clusters by classes of data. Furthermore, the output layer introduces inhibitory connections whose level of inhibition increases in proportion to the square root of the Euclidean distance between neurons [15].

V. Experiments

In this section, we introduce the RAVEN dataset and implementation details of both our supervised and unsupervised approaches.

A. Dataset

To demonstrate the effectiveness of our methods, we evaluate them on the latest RAVEN dataset [14]. In total, the RAVEN dataset consists of $1,120,000$ images and $70,000$ RPM problems, equally distributed in 7 distinct figure configurations: `Center`, `2x2Grid`, `3x3Grid`, `Left-Right` (`L-R`), `Up-Down` (`U-D`), `Out-InCenter` (`O-IC`), and `Out-InGrid` (`O-IG`) as shown in Figure 3. In addition, there is an average of 6.29 rules for each problem. To evaluate the reasoning ability of machines, the RAVEN dataset also contains the results of human-level performance. More details of the RAVEN dataset are described in [14].

B. Implementation Details

To simulate our SNNs we use Python, BindsNET [21], an NVIDIA Tesla T4 16GB GPU and an Intel Xeon Platinum 8259CL 16 Cores 2.50GHz CPU. Table I summarises the run times for the majority of our experiments, from which we can observe that using extracted features rather than the raw dataset provides a $250\times$ speed-up. For all our experiments, besides using extracted features, we still follow the same experimental format in [14]. Namely, the RAVEN dataset is split into three parts, 6 folds for training, 2 for validation and the rest 2 for testing. We train our models on the training set, tune the model parameters on the validation set and report accuracy on the test set.

For our feature extraction method, we use a mini-batch of 50 and an Adam optimiser with learning rate 10^{-4}, and due to the standard image resolution in ResNet [22], all images in the problem matrix and answer set are resized to a fixed size of 224×224. Besides resizing the images for ResNet, we do not manipulate the dataset or use use any data augmentation methods to manipulate the diversity of our training samples, this is to ensure that all our comparisons are as fair as possible.

TABLE I: Elapsed real time comparisons for each of our models for the two major types of experiments we conducted. The networks are trained and tested on either extracted features or the raw dataset.

Baseline Experiments		
Model	Extracted Features	Raw Dataset
MLP	4 minutes	16 hours
LSM	3.35 hours	839 hours
SNN	4.15 hours	1038 hours
SSOM	4.34 hours	1085 hours
Generalization Experiments		
MLP	1 minute	4 hours
LSM	29 minutes	121 hours
SSOM	36 minutes	150 hours
SNN	37 minutes	154 hours

For our LSM, we use a mini-batch of 1 with the STDP learning rule and a SGD optimiser with momentum to learn network parameters in the liquid and output layer, respectively. For the hyper parameters of our LSM, we have a threshold value v_θ of -52.0; a reset potential v_r of -65.0; an absolute refractory period Δ_{abs} of 5 milliseconds; a voltage decay constant τ_v of 0.99; and a simulation time step δ_t of 1.0 millisecond. For the SGD optimiser we use a learning rate of 10^{-4} and a momentum value of 0.9.

For our SNN, we also use a mini-batch of 1, and to learn network parameters, we use the STDP learning rule between the input and output layer, and a hybrid STDP and SOM learning algorithm for the recurrent connections in the output layer. For hyper parameters, we use the same as those utilised for the LSM with the addition of hyper parameter c_{inhib} which is multiplied by the distance of the neurons to compute inhibition levels, and c_{max}, which is utilised to specify the maximum allowed inhibition. For their values, we use 1.0 and 100.0 for c_{inhib} and c_{max}, respectively.

VI. Results

In this section, we report the performance of our approaches and compare them with previous reports in the literature.

A. Comparison with Baselines

To demonstrate the effectiveness of our method, we report all available results on the RAVEN dataset for comparison, which include LSTM, CNN, ResNet, WReN and DRT. Detailed implementations of these baselines are described in [14]. In addition, we compare our models to more recent approaches, these include CoPINet [6], ResNet-50 [7] and MXGNet [17].

Table II shows the testing accuracy of each supervised learning model on the RAVEN dataset. We can observe that the performance for solving RPMs heavily relies on the network architecture. LSTM and WReN perform poorly on this dataset, with only being slightly better than random guessing (12.50%).

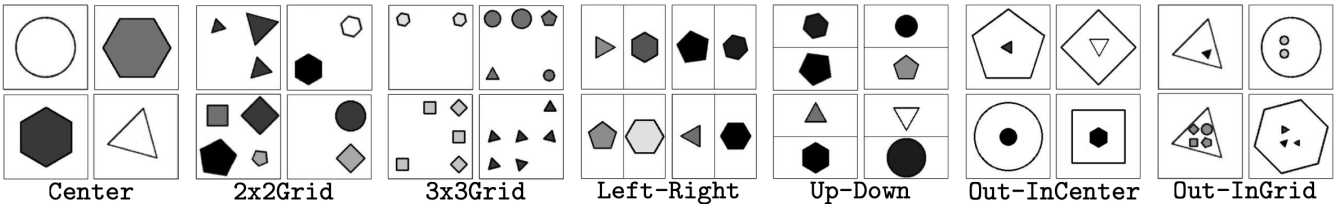

Fig. 3: Examples of 7 different figure configurations in the RAVEN dataset.

TABLE II: Testing accuracies for different networks using supervised learning.

Network	Testing Accuracy (%)
Random	12.50
LSTM [14]	13.07
WReN [14]	14.69
CNN [14]	36.97
ResNet-18+MLP [14]	53.43
ResNet-18 [7]	77.18
LSTM+DRT [14]	13.96
WReN+DRT [14]	15.02
CNN+DRT [14]	39.42
ResNet-18+MLP+DRT [14]	59.56
MXGNet [17]	83.91
ResNet-50 [7]	86.26
LSM (ours)	88.20
CoPINet [6]	91.42
LSM+DRT (ours)	**93.30**
Human	84.41

TABLE III: Testing accuracies for different networks using unsupervised learning.

Network	Testing Accuracy (%)
Random	12.50
MCPT (one-hot) [7]	20.55
MCPT (two-hot) [7]	28.50
SNN	54.15
SSOM	71.04
SNN+DRT	84.86
SSOM+DRT	**93.23**

The CNN model obtains an accuracy of 36.97%, which is still poor. By using a DRT module with a ResNet backbone, ResNet-18+MLP+DRT obtains an increase in performance of about 12% when compared to ResNet18+MLP. In contrast, ResNet-18 from [7] outperforms ResNet18+MLP+DRT by a large margin of 17.62% and achieves a testing accuracy of 77.18%. A potential reason for such a result is that replacing the original 1000 fully connected layer in ResNet-18 with two 512 fully connected layers may reduce its testing accuracy.

In Table II, we can also observe that only four of fourteen models surpass human-level performance, i.e., our LSM+DRT (93.30%) model, the CoPINet model (91.42%), our standard LSM model (88.20%) and the ResNet-50 model (86.26%). The average percent difference between our best performing model and the CoPINet model is about 2%. However note that it was not possible to perform a standard statistical test on this difference due to three reasons: (1) only one result is available for CoPINet; (2) computational costs prevented us from doing multiple runs with ResNet, i.e., we could only do 30 runs with the LSM+DRT model; and (3) the results from the multiple runs were not normally distributed, having a Kolmogorov-Smirnov test probability of 0.264. Nevertheless all 30 runs generated higher accuracy than CoPINet, with a corresponding p-value of $< 10^{-9}$.

From Table II we can finally observe that our standard LSM model outperforms the ResNet-50 model without the need of any extra pre-training on a large image dataset such as ImageNet [7]. However, also notice that the best performing models in Table II use additional network design, which further emphasises the observation that solving RPMs may heavily rely on network architecture and deep layers.

For our spiking neural network, we take a similar approach in reporting results as with our liquid state machine. We compare our spiking neural network and its variants in Table III with the only other previous unsupervised results on the RAVEN dataset, i.e., the MCPT approach by [7]. From Table III we can observe that the MCPT approach, either the one-hot or two-hot variant, performs poorly. The one-hot variant only gets a performance slightly above random guessing with an accuracy of 20.55%, while the two-hot variant slightly doubles random guessing with an accuracy of 28.50%. When compared to our worst performing unsupervised model (SNN), we can observe that our model outperforms the MCPT two-hot model with a percent difference of about 56.38%.

Following a similar trend to the supervised methods, we can observe in Table III that network architecture also plays an important role in solving RPMs with unsupervised methods. For instance, we can observe that using features from the DRT module with our basic spiking model increases accuracy by about 56.71%. We can also observe that hybridizing the STDP learning rule with a SOM algorithm increases accuracy by about 31.23%. This last observation is especially interesting as it illustrates the importance of learning algorithms in solving RPMs, at least with unsupervised methods.

From Table III we can observe that only two models achieve or outperform human performance on the RAVEN dataset, i.e., our SNN+DRT model (84.86%) and our SSOM+DRT model (93.23%). However, we can observe that unsupervised methods currently only achieve human-level performance by relying on a module that incorporates domain knowledge. If we were to compare the best performing "naive" unsupervised model, i.e., a model which does not rely on any domain knowledge, we can observe that there is a 13.37% difference between that model (SSOM) and a human evaluated on the RAVEN dataset. This observation clearly indicates that our SSOM model has closed the gap between unsupervised learning and human performance on the RAVEN dataset, but there is plenty of room for improvement.

Finally, from Tables II and III we can observe that using

TABLE IV: Generalization test. The networks are trained on `Center` and tested on three other figure configurations.

Network	`Center`	`L-R`	`U-D`	`O-IC`
DRT [14]	51.87	40.03	35.46	38.84
ResNet-50 [7]	60.80	43.65	41.40	43.65
LSM	86.90	**87.95**	**87.90**	87.45
LSM+DRT	**87.45**	87.90	87.80	**88.70**
SNN	57.55	51.75	49.00	52.25
SNN+DRT	58.64	48.40	48.90	52.05
SSOM	69.90	50.05	49.30	53.85
SSOM+DRT	84.60	70.03	65.15	67.55

networks with recurrent spiking modules greatly improves performance in solving RPMs. One possible explanation for this is that spiking neurons can successfully perform multi-category classifications by responding to each category with unique output spike patterns. Furthermore, spiking neurons can be sensitive to afferent spike timings, such that their output spike patterns extract and encode temporal features without any explicit instruction [23], [24]. In the context of solving RPMs, we argue that these two properties alone clearly give spiking modules an advantage.

B. Generalization Test

To measure how well our models trained on one figure configuration and tested on other similar configurations, we further test the generalization ability of our approaches on the RAVEN dataset. Similarly to the experimental format in [14], we evaluate our models with three kinds of configurations. The first generalization experiment involves training the models on `Center` and testing them on `L-R`, `U-D` and `O-IC`. The second experiment involves training on `L-R`, and testing on `U-D`, and vice versa. While the third experiment involves training on `2x2Grid` and testing on `3x3Grid`, and vice versa. Finally, for our last experiment, we evaluate the performance of our models with fewer training samples.

The first experiment attempts to measure the compositional reasoning abilities of the models, as it requires them to learn the rules from RPMs with single-component configurations, and generalise them to RPMs with multiple independent but similar components. In Table IV, we can observe that our SNN, SSOM and our LSM models outperform ResNet-50 [7] and ResNet18+MLP+DRT [14] (denoted as DRT) on generalization. We can also observe that except for our LSM model, it is difficult for the other models to generalise to more complex RPMs after being trained on simpler cases, i.e., generalizing from `Center` to `L-R`, `U-D` or `O-IC`.

The second and third experiments respectively attempt to measure the transposition capabilities of the models and their generalization capabilities when the number of objects change. From Table V we can observe that our LSM model, when not trained on DRT features, outperforms all other models on transposition. However interestingly, when trained with DRT features, the performance of the LSM model slightly declines on transposition. A potential explanation for this is that the DRT module may have slight biases for grid structured RPMs. This explanation is further emphasised when we observe the performance of the LSM model on `2x2Grid` and `3x3Grid` configurations. Namely, in Table V, we can observe that the LSM model outperforms all other models on object number generalization, and attains high consistent performance when trained with DRT extracted features.

Additionally, in Table V we can also observe that most models struggle with generalization when they are trained on simple cases but tested on complicated ones. In particular, we can observe in Table V that most models attain slightly lower accuracies when trained on `2x2Grid` and tested on `3x3Grid` configurations. One exception to this observation is the SNN model when trained with ResNet extracted features only, much like humans the SNN model is able to solve complex problems after training on simpler ones, however unlike humans, its performance level is still relatively low.

Finally, in Table VI we can observe the results from our last experiment. This experiment attempts to reduce the inherent bias towards our models when comparing their performance to those of human subjects. Namely, since human subjects never experience the same intensive training as our models, we consider any comparison between the two to be inherently unfair. Hence, to reduce this bias and make the comparison fairer, we report the performance of our models with fewer training samples. In Table VI we can observe that our LSM model, either trained with or without features extracted from the DRT module, manages to achieve or surpass human-level performance. More interestingly, with about $1/64$th of the training samples, our LSM model surpasses human-level performance when trained with DRT extracted features and does so with about $1/30$th of the training samples when it is not. Meanwhile, with about $1/8$th of the training samples, our LSM model easily achieves comparable accuracy to CoPINet, when trained with features extracted from the DRT module.

From Table VI we can observe that a major advantage of using spiking modules is their ability to learn a reasonably good data representation while seeing only a small number of examples. This observation is further emphasised when we consider that our SNN models, either trained with or without features from the DRT module, are asymptotic in accuracy with about $1/30$th of the training samples. Finally, the results from Table VI also indicate that spiking modules are able to develop appropriate filters quickly and gradually refine these filters with increasing examples of training data.

VII. Conclusion

In this work, we attempt to solve RPMs with spiking modules, and investigate their performance with either supervised or unsupervised learning. We perform extensive experiments on the RAVEN dataset to verify the effectiveness of our approaches and demonstrate how they significantly outperform existing approaches. Furthermore, we show the advantage of using biologically inspired modules, their inherent representational power and their ability to solve analogical reasoning problems. More importantly, we show that unlike their non-spiking counterparts, spiking modules are able to extract and encode temporal features without any explicit instruction, do not heavily rely on training data, and are more readily able to generalize easier to new problems. From our various

TABLE V: Generalization test. The networks are trained on `Left-Right` and tested on `Up-Down`, and vice versa. Additionally, the networks are trained on `2x2Grid` and tested on `3x3Grid`, and vice versa.

Config	DRT [14]		ResNet-50 [7]		SNN		SNN+DRT		SSOM		SSOM+DRT		LSM		LSM+DRT	
	L-R	U-D	L-R	U-D	L-R	U-D	L-R	U-D	L-R	U-D	L-R	U-D	L-R	U-D	L-R	U-D
L-R	41.07	38.10	60.95	57.15	51.30	46.40	41.20	48.55	74.75	75.80	75.60	76.75	**86.95**	**88.60**	86.90	87.90
U-D	39.48	43.60	59.20	63.75	53.50	57.05	44.65	42.15	73.45	70.10	81.80	74.65	**88.20**	**88.25**	86.90	87.90
	2x2	3x3	2x2	3x3	2x2	3x3	2x2	3x3	2x2	3x3	2x2	3x3	2x2	3x3	2x2	3x3
2x2	40.93	38.69	35.90	35.55	50.55	52.30	50.70	49.60	64.70	62.90	81.55	80.10	87.20	84.55	**88.80**	**87.95**
3x3	40.93	38.69	35.90	35.55	52.05	53.95	54.45	54.25	71.40	69.25	82.95	82.45	88.65	87.15	**88.80**	**87.95**

TABLE VI: Model performance under different training set sizes, while test set size remains unchanged. The full training set has $42,000$ samples.

	Accuracy (%)						
No. of Samples	CoPINet [6]	SNN	SNN+DRT	SSOM	SSOM+DRT	LSM	LSM+DRT
658	44.48	47.04	54.19	47.94	60.90	71.01	**87.51**
$1,316$	57.69	50.10	63.64	49.31	70.27	85.67	**88.72**
$2,625$	65.55	50.86	68.40	53.83	74.32	86.64	**91.14**
$5,250$	74.53	51.02	73.41	55.89	80.27	87.74	**92.21**
$10,500$	80.92	54.60	78.76	56.02	84.24	87.84	**93.26**
$21,000$	86.43	57.21	80.81	61.18	88.75	88.04	**93.30**

experiments, we hope that our results inspire further research into spiking modules and solving RPM problems.

REFERENCES

[1] P. A. Carpenter, M. A. Just, and P. Shell, "What one intelligence test measures: a theoretical account of the processing in the raven progressive matrices test." *Psychological review*, vol. 97, no. 3, p. 404, 1990.

[2] J. Raven, "The raven's progressive matrices: change and stability over culture and time," *Cognitive psychology*, vol. 41, no. 1, pp. 1–48, 2000.

[3] G. Domino and M. L. Domino, *Psychological testing: An introduction*. Cambridge University Press, 2006.

[4] A. Voulodimos, N. Doulamis, A. Doulamis, and E. Protopapadakis, "Deep learning for computer vision: A brief review," *Computational intelligence and neuroscience*, vol. 2018, 2018.

[5] A. Santoro, F. Hill, D. Barrett, A. Morcos, and T. Lillicrap, "Measuring abstract reasoning in neural networks," in *International Conference on Machine Learning*, 2018, pp. 4477–4486.

[6] C. Zhang, B. Jia, F. Gao, Y. Zhu, H. Lu, and S.-C. Zhu, "Learning perceptual inference by contrasting," in *Advances in Neural Information Processing Systems*, 2019, pp. 1075–1087.

[7] T. Zhuo and M. Kankanhalli, "Solving raven's progressive matrices with neural networks," *arXiv preprint arXiv:2002.01646*, 2020.

[8] D. Liu and S. Yue, "Event-driven continuous stdp learning with deep structure for visual pattern recognition," *IEEE transactions on cybernetics*, vol. 49, no. 4, pp. 1377–1390, 2018.

[9] M. Oquab, L. Bottou, I. Laptev, and J. Sivic, "Learning and transferring mid-level image representations using convolutional neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 1717–1724.

[10] C. F. Cadieu, H. Hong, D. L. Yamins, N. Pinto, D. Ardila, E. A. Solomon, N. J. Majaj, and J. J. DiCarlo, "Deep neural networks rival the representation of primate it cortex for core visual object recognition," *PLoS Comput Biol*, vol. 10, no. 12, p. e1003963, 2014.

[11] D. C. Krawczyk, "The cognition and neuroscience of relational reasoning," *Brain research*, vol. 1428, pp. 13–23, 2012.

[12] W. Maass, T. Natschläger, and H. Markram, "Real-time computing without stable states: A new framework for neural computation based on perturbations," *Neural computation*, vol. 14, no. 11, pp. 2531–2560, 2002.

[13] N. Soures and D. Kudithipudi, "Deep liquid state machines with neural plasticity for video activity recognition," *Frontiers in neuroscience*, vol. 13, p. 686, 2019.

[14] C. Zhang, F. Gao, B. Jia, Y. Zhu, and S.-C. Zhu, "Raven: A dataset for relational and analogical visual reasoning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 5317–5327.

[15] H. Hazan, D. Saunders, D. T. Sanghavi, H. Siegelmann, and R. Kozma, "Unsupervised learning with self-organizing spiking neural networks," in *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2018, pp. 1–6.

[16] X. Steenbrugge, S. Leroux, T. Verbelen, and B. Dhoedt, "Improving generalization for abstract reasoning tasks using disentangled feature representations," *arXiv preprint arXiv:1811.04784*, 2018.

[17] D. Wang, M. Jamnik, and P. Lio, "Abstract diagrammatic reasoning with multiplex graph networks," *arXiv preprint arXiv:2006.11197*, 2020.

[18] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[19] W. Gerstner and W. M. Kistler, *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press, 2002.

[20] P. U. Diehl and M. Cook, "Unsupervised learning of digit recognition using spike-timing-dependent plasticity," *Frontiers in computational neuroscience*, vol. 9, p. 99, 2015.

[21] H. Hazan, D. J. Saunders, H. Khan, D. Patel, D. T. Sanghavi, H. T. Siegelmann, and R. Kozma, "Bindsnet: A machine learning-oriented spiking neural networks library in python," *Frontiers in neuroinformatics*, vol. 12, p. 89, 2018.

[22] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[23] S. Guo, Z. Yu, F. Deng, X. Hu, and F. Chen, "Hierarchical bayesian inference and learning in spiking neural networks," *IEEE transactions on cybernetics*, vol. 49, no. 1, pp. 133–145, 2017.

[24] Q. Yu, H. Li, and K. C. Tan, "Spike timing or rate? neurons learn to make decisions for both through threshold-driven plasticity," *IEEE transactions on cybernetics*, vol. 49, no. 6, pp. 2178–2189, 2018.