A Closer Look at Generalisation in RAVEN: Appendices

Steven Spratley, Kris Ehinger, and Tim Miller

School of Computing and Information Systems
The University of Melbourne, Victoria, Australia

A. Context-blind Performance

We list the results of context-blind variants of our solvers in Table 1. Note that these solvers have the stack function removed; sequence encoders are simply given all answer frames/embeddings instead. Rel-Base displays near-perfect performance in several configurations; this, as mentioned in our main paper, points to the importance of independent processing in comparing answer frames over RAVEN, if it is to be used to fairly assess the reasoning ability of these networks.

Table 1. Accuracy (%) of ResNet and Rel-Base, trained context-blind on RAVEN.

	Acc	Centre	2x2	3x3	L-R	U-D	O-IC	O-IG
ResNet	83.11	84.23	65.34	68.70	95.14	95.82	92.02	80.53
Rel-Base	92.46	98.49	78.66	80.52	99.22	99.66	98.63	92.04

B. Model Parameters

In Table 2, we provide all hyperparameters essential to reproducing the models in our main paper. This is intended to complement our code, released online[1]. All models were optimised with Adam, using *de facto* default parameters: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$, and a learning rate of $3e{-}4$. We used split-batch training to enable parallelisation across two GPUs, with a batch size of 32.

We wish to refer readers to the online repository of the Pyro library[2] as the hyperparameters set by this code were largely unchanged in our AIR module; we used a learning rate of 0.1 for the data-dependent baselines, a z presence prior of 0.01, and a decoder output bias of -2. Where we differed was in the scale prior's mean and standard deviation, μ and σ, which had to be fine-tuned for specific problem configurations in RAVEN. For the most part, we found $\mu = 2.0$ and $\sigma = 0.4$ worked well. For the `3x3Grid` set, pushing μ to 3.0 and σ to 0.2 was necessary to predispose the module to using small, regularly-sized attention windows – and therefore fill all 9 slots, instead of perceiving larger compound objects. As mentioned in the paper, AIR didn't correctly decompose `Out-InGrid`; instead, it produced 2 slots corresponding to the outer and grouped inner shapes.

[1] `https://github.com/SvenShade/Rel-AIR`
[2] `https://github.com/pyro-ppl/pyro/tree/dev/examples/air`

Table 2. Details of the modules built for ResNet, Rel-Base, and Rel-AIR, along with their components listed in architectural order. Note that 'in/hidden/out' refers to the number of channels, and N refers to the number of object slots.

Module	Component	Parameters
Convolutional block	Convolutional layer (1D or 2D)	Kernel size: 7 Padding: 3 Stride: 1 if 1D layer 2 if 2D layer
	ELU nonlinearity	
	Batch normalisation	
	Spatial dropout	Probability: 0.1
Residual block	Convolutional block	
	Convolutional block	
	Skip connection	+ max. pool if strided blocks
Frame encoder (Rel-Base) & Object encoder (Rel-AIR)	2D Residual block	In/hidden/out: 1, 64, 64
	2D Residual block	In/hidden/out: 64, 64, 16
Frame encoder (Rel-AIR)	1D Residual block	In/hidden/out: N, 128, 128
	1D Residual block	In/hidden/out: 128, 128, 1
Frame conditioning (Rel-AIR)	Bilinear layer	In-1/In-2/out: 400, 3, 403
	ELU nonlinearity	
Sequence encoder (Rel-Base & Rel-AIR)	1D Residual block	In/hidden/out: 9, 64, 128
	Max. pool	Downsample: 4x
	1D Residual block	In/hidden/out: 128, 128, 64
	Adaptive avg. pool	Size: 16
Sequence encoder (ResNet)	2D Residual block	In/hidden/out: 9, 64, 64
	2D Residual block	In/hidden/out: 64, 64, 64
MLP (All models)	Linear layer	In: 1600 (ResNet) 1024 (Rel-Base & Rel-AIR) Out: 512
	ELU nonlinearity	
	Batch normalisation	
	Dropout	Probability: 0.5
	Linear layer	In/out: 512, 1