7. Appendix

7.1. Project's webpage

Full details of the project (including video results, codebase, etc) are available at https://sites.google.com/view/abstractions4rl.

7.2. Overview of all methods used in baselines and ablations

The environmental setting and the feature extractor used in all the variations and different methods considered is summarized in Table 7.2

	Observation			Latent Features	Demos	Rewards
	Vision (RGB)	Joint Encoders	Environment State			
RRL(Ours)	✓	✓		Resnet34	✓	Sparse
RRL(Resnet18)	✓	✓		Resnet18	✓	Sparse
RRL(Resnet50)	✓	✓		Resnet50	✓	Sparse
RRL (VAE)	✓	✓		VAE	✓	Sparse
RRL(Vision)	✓			Resnet34	✓	Sparse
FERM	✓	✓			✓	Sparse
NPG(State)		✓	✓			Sparse
NPG(Vision)	✓			Resnet34		Sparse
DAPG(State)		✓	✓		✓	Sparse
RRL(Sparse)	✓	✓		Resnet34	✓	Sparse
RRL(Dense)	✓	✓		Resnet34	✓	Dense
RRL(Noise)	✓	✓		Resnet34	✓	Sparse
RRL(Vision + Sensors)	✓	✓		Resnet34	✓	Sparse
RRL(ShuffleNet)	✓	✓		ShuffleNet-v2	✓	Sparse
RRL(MobileNet)	✓	✓		MobileNet-v2	✓	Sparse
RRL(vdvae)	✓	✓		Very Deep VAE	✓	Sparse

7.3. RRL(Ours)

Parameters	**Setting**
BC batch size	32
BC epochs	5
BC learning rate	0.001
Policy Size	(256, 256)
vf _batch_size	64
vf_epochs	2
rl_step_size	0.05
rl_gamma	0.995
rl_gae	0.97
lam_0	0.01
lam_1	0.95

Table 2. **Hyperparameter details for all the RRL variations.**

Same parameters are used across all the tasks (Pen, Door, Hammer, Relocate, PegInsertion, Reacher) unless explicitly mentioned. Sparse reward setting is used in all the hand manipulation environments as proposed by **?** along with 25 expert demonstrations. We have directly used the parameters (summarize in Table 2) provided by DAPG without any additional hyperparameter tuning except for the policy size (used same across all tasks). On the Adroit Manipulation task,

200 trajectories for Hammer-v0, Door-v0, Relocate-v0 whereas 400 trajectories for Pen-v0 per iteration are collected in each iteration.

7.4. Results on MJRL Environment

We benchmark the performance of RRL on two of the MJRL environments (Rajeswaran et al., 2020), Reacher and Peg Insertion in Figure 10. These environments are quite low dimensional (7DoF Robotic arm) compared to the Adroit hand (24 DoF) but still require rich understanding of the task. In the peg insertion task, RRL delivers state comparable (DAPG(State)) results and significantly outperforms FERM. However, in the Reacher task, we notice that DAPG(State) and FERM perform surprisingly well whereas RRL struggles to perform initially. This highlights that using task specific representations in simple, low dimensional environments might be beneficial as it is easy to overfit the feature encoder for the task in hand while the Resnet features are quite generic. For the MJRL environment, shaped reward setting is used as provided in the repository [2] along with 200 expert demonstrations. For the Peg Insertion task 200 trajectories and for Reacher task 400 trajectories are collected per iteration.

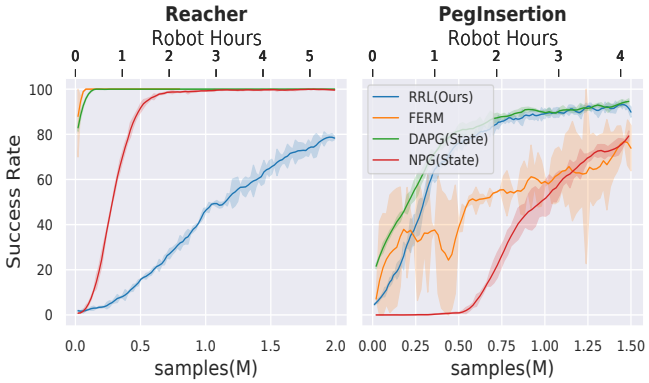

Figure 10. **Results on MJRL Environment.** RRL outperforms FERM and delivers results on par with DAPG(State) in the PegInsertion task. In Reacher, FERM outperforms RRL following that learning task specific representations is easier in simple tasks.

7.5. Other variations of RRL

a) **RRL(MobileNet), RRL(ShuffleNet)** : The encoders (ShuffleNet (Ma et al., 2018) and MobileNet (Sandler et al., 2019)) are pretrained on ImageNet Dataset using a classification objective. We pick the pretrained models from torchvision directly and freeze the parameters during the entire training of the RL agent. Similar to RRL(Ours), the last layer of the model is removed and a latent feature of dimension 1024 and 1280 in case of ShuffleNet and MobileNet respectively is used.
b) **RRL(vdvae)** : We use a very recent state of the art hierarchical VAE (Child, 2021) that is trained on ImageNet dataset. The code along with the pretrained weights are publically available [3]. We use the intermediate features of the encoder of dimension 512. All the parameters are frozen similar to RRL(Ours).

7.6. DMControl Experiment Details

For the RAD (Laskin et al., 2020), CURL (Srinivas et al., 2020), SAC+AE (Yarats et al., 2020) and State SAC (Haarnoja et al., 2019), we report the numbers directly provided by Laskin et al.. For SAC+RRL, Resnet34 is used as a fixed feature extractor and the past three output features (frame_stack$=3$) are used as a representative of state information in SAC algorithm. For the fixed RAD encoder, we train a the RL agent along with RAD encoder using the default hyperparameters provided by the authors for Cartpole environment. We used the trained encoder as a fixed feature extractor and retrain the policies using the same hyperparameters (except for the frame_skip or action_repeat) for all the tasks. The frame_skip values are task specific as mentioned in (Yarats et al., 2020) and mentioned in Table 4. The hyperparameters used are summarized in the table 3. SAC implementation in PyTorch courtesy (Yarats & Kostrikov, 2020).

7.7. RRL(VAE)

For training, we collected a dataset of 1 million images of size 64 x 64. Out of the 1 million images collected, 25% of the images are collected using an optimal course of actions (expert policy), 25% with a little noise (expert policy + small noise), 25% with even higher level of noise (expert policy + large noise) and remaining portion by randomly sampling actions (random actions). This is to ensure that the images collected sufficiently represents the distribution faced by policy during the training of the agent. We observed that this significantly helps compared to collecting data only from the expert policy.

[2] https://github.com/aravindr93/mjrl
[3] https://github.com/openai/vdvae

Parameter	Setting
frame_stack	3
replay_buffer_capacity	100000
init_steps	1000
batch_size	128
hidden_dim	1024
critic_lr	1e-3
critic_beta	0.9
critic_tau	0.01
critic_target_update_freq	2
actor_lr	1e-3
actor_beta	0.9
actor_log_std_min	-10
actor_log_std_max	2
actor_update_freq	2
discount	0.99
init_temperature	0.1
alpha_lr	1e-4
alpha_beta	0.5

Table 3. SAC hyperparameters.

Environment	action_repeat
Cartpole, Swing	8
Reacher, Easy	4
Cheetah, Run	4
Cup, Catch	4
Walker, Walk	2
Finger, Spin	2

Table 4. Action Repeat Values for DMControl Suite

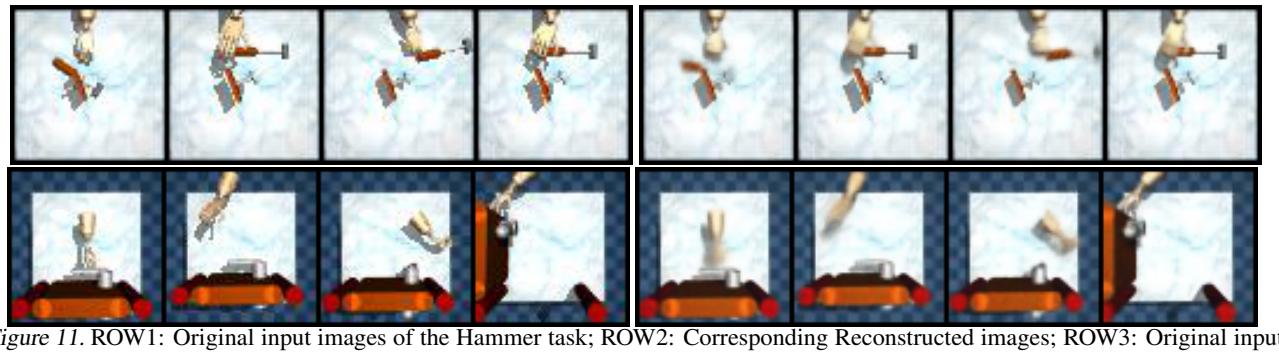

Figure 11. ROW1: Original input images of the Hammer task; ROW2: Corresponding Reconstructed images; ROW3: Original input images of the Door task; ROW4: Corresponding Reconstructed images. These images depict that the latent features sufficiently encodes features required to reconstruct the images.

The variational auto-encoder(VAE) is trained using a reconstruction objective (Kingma & Welling, 2014) for 10epochs. Figure 11 showcases the reconstructed images. We used a latent size of 512 for a fair comparison with Resnet. The weights of the encoder are freezed and used as feature extractors in place of Resnet in RRL. RRL(VAE) also uses the inputs from the pro-prioceptive sensors along with the encoded features. VAE implementation courtesy (Subramanian, 2020).

7.8. Visual Distractor Evaluation details

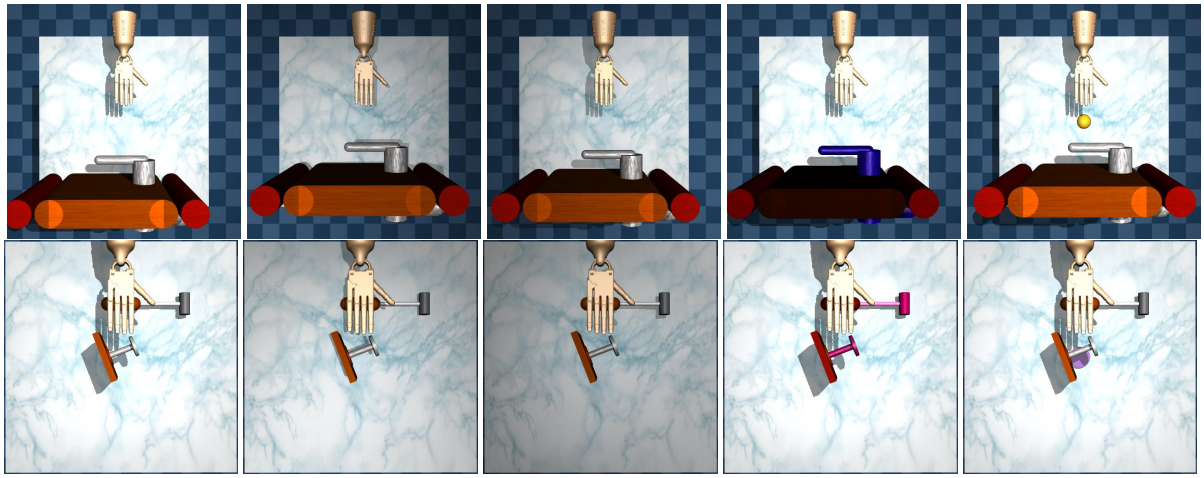

Figure 12. COL1: Original images; COL2: Change in light position; COL3: Change in light direction; COL4: Randomizing object colors; COL5: Introducing a random object in the scene. All the parameters are randomly sampled every time in an episode.

In order to test the generalisation performance of RRL and FERM (Zhan et al., 2020), we subject the environment to various kinds of visual distractions during inference (Figure 12). Note all parameters are freezed during this evaluation, an average performance over 75 rollouts is reported. Following distractors were used during inference to test robustness of the final policy -

- Random change in light position.
- Random change in light direction.
- Random object color. (Handle, door color for Door-v0; Different hammer parts and nail for Hammer-v0)
- Introducing a new object in scene - random color, position, size and geometry (Sphere, Capsule, Ellipsoid, Cylinder, Box).

7.9. Compute Cost calculation

We calculate the actual compute cost involved for all the methods (RRL(Ours), FERM, RRL(Resnet-50), RRL(Resnet-18)) that we have considered. Since in a real-world scenario there is no simulation of the environment we do not include the

cost of simulation into the calculation. For fair comparison we show the compute cost with same sample complexity (4 million steps) for all the methods. FERM is quite compute intensive (almost 3x RRL(Ours)) because (a) Data augmentation is applied at every step (b) The parameters of Actor and Critic are updated once/twice at every step (Compute results shown are with one update per step) whereas most of the computation of RRL goes in the encoding of features using Resnet. The cost of VAE pretraining in included in the over all cost. RRL(Ours) that uses Resnet-34 strikes a balance between the computational cost and performance. **Note:** No parallel processing is used while calculating the cost.