IMAGE AUGMENTATION AND AUXILIARY LOSS DUO

Faraaz Mallick, Dewang Modi, and Amatya Sharma

IIT Kharagpur
Computer Science and Engineering

1. INTRODUCTION

Sample-efficient deep reinforcement learning (RL) algorithms capable of directly training from image pixels would open up many real-world applications in control and robotics. However, simultaneously training a convolutional encoder alongside a policy network is challenging when given limited environment interaction, strong correlation between samples and a typically sparse reward signal. There have been naive attempts to use a large capacity encoder which resulted in severe over-fitting and smaller encoders produce impoverished representations that limit task performance.

However, training an agent to solve control tasks directly from high-dimensional images with model-free reinforcement learning (RL) has proven difficult. Instead a promising approach is to learn a latent representation (z_t) together with the control policy. However, even in this case fitting a high-capacity encoder using a scarce reward signal is sample inefficient and leads to poor performance. Prior work has shown that auxiliary losses, such as image reconstruction loss (i.e. using a decoder to recreate the image and track the difference between input and output images), weight regularization, noise injection[1], or various forms of auto-encoder[2] can aid efficient representation learning. In additional to this RL, reconstruction objectives[3] or alternate tasks are often used [4]. Though, these objectives alone are unrelated to the task at hand, thus have no guarantee of inducing an appropriate representation for the policy network. Another drawback is that incorporating reconstruction loss into an off-policy learning algorithm often leads to training instability. However, Yarats et al. [5] explored the underlying reasons of this training instaility and identified variational autoencoders, used by previous investigations, as the cause of the divergence. Following these findings, they proposed effective techniques (SAC+AC) to improve training stability. This resulted in a simple approach capable of matching state-of-the-art model-free and model-based algorithms on MuJoCo control tasks. Furthermore, their approach demonstrated robustness to observational noise, surpassing existing approaches in this setting.

Data augmentation methods have proven highly effective in vision and speech domains, where output-invariant perturbations can easily be applied to the labeled input examples. Surprisingly, data augmentation has received relatively little attention in the RL community. Outperforming the previous work [5], recently, it has been shown in [6] that data augmentation techniques, commonplace in Computer Vision but not in RL, are also important for achieving the state-of-the-art performance in image-based RL.

In this paper, we first explore the reproducibility of SAC-AE algorithm given in [5]. With optimal hyperparameter settings, the model is quite slow and takes excessive amount of memory as well as time to get reach optimal reward values even with a machine setup of 104GB RAM and 1xTesla T4 GPU. We scrutinize the code to check the logical equivalence with the proposed algorithms in the paper, and run for 16,000 training steps on the aforementioned machine (took around 18 hours) to verify the code and mention the errors and observations in Section 4.1.

Secondly, we implement our model ($\mathcal{AD}$) with the central idea to combine the two aforementioned ideas of (1) Data Augmentation (based on the work in [6; 7]), and (2) Auxiliary losses, in particular Image Reconstruction (based on the work in [5]). We aim is to use standard image transformations to perturb input observations, as well as incorporate image construction loss by adding a decoder to the Auto-encoder, so that different transformations of the same input image have similar Q-function values. We run our experiments on the four possible variations of $\mathcal{AD}$-model ($\mathcal{AD}$(0,0), $\mathcal{AD}$(0,1), $\mathcal{AD}$(1,0), $\mathcal{AD}$(1,1)) by adding/removing data-augmentation and reconstruction-losses. Moreover, we employ DDPG as employed by [7] (first defined in [8]). This is much faster compared to SAC and we compare all the 4 variations of $\mathcal{AD}$-model by running over 100,000 training steps. We record our observations and possible explanations in Section 4.2.

2. RELATED WORKS

Efficient learning from high-dimensional pixel observations has been a problem of paramount importance for model-free RL. While some impressive progress has been made applying model-free RL to domains with simple dynamics and dis-

Authors are listed alphabetically (Equal contributions). The implementation of our code and evaluation videos are available at https://github.com/dewangmodi/Image-Augmentation-And-Auxiliary-Loss-Duo crete action spaces [9], attempts to scale these approaches to complex continuous control environments have largely been unsuccessful, both in simulation and the real world. A glaring issue is that the RL signal is much sparser than in supervised learning, which leads to sample inefficiency, and higher dimensional observation spaces such as pixels worsens this problem.

Now we take a look at the recent work that has been done in the creation and improvement of sample-efficient continuous control methods that observe high-dimensional images. All of the work has been done by Yarats et al.

2.1. SAC+AE

In [5], the authors first try to understand why previous approaches of adding an autoencoder to model-free RL approaches, with a focus on off-policy methods, did not work, they then confirm the vitality of pixel reconstruction loss for learning a good representation, specifically when trained jointly. Based on these findings, they recommended a simple and effective autoencoder-based off-policy method that can be trained, and called the model, **SAC+AE**.

The model is based on the Soft Actor-Critic model, which uses the maximum entropy framework. At each iteration SAC performs soft policy evaluation and improvement steps. The policy evaluation step fits a parametric Q-function $Q(\mathbf{s}_t, \mathbf{a}_t)$ using transitions sampled from the replay buffer D by minimizing the soft Bellman residual

$$J(Q) = \mathbb{E}_{(\mathbf{s}_t,\mathbf{a}_t,r_t,\mathbf{s}_{t+1})\sim D}\left[\left(Q(\mathbf{s}_t,\mathbf{a}_t) - r_t - \gamma\bar{V}(\mathbf{s}_{t+1})\right)^2\right]$$

The target value function $\bar{V}$ is approximated via a Monte Carlo estimate of the following expectation

$$\bar{V}(\mathbf{s}_t) = \mathbb{E}_{\mathbf{a}_t\sim\pi}\left[\bar{Q}(\mathbf{s}_t,\mathbf{a}_t) - \alpha\log\pi(\mathbf{a}_t|\mathbf{s}_t)\right]$$

where $\bar{Q}$ is the target Q-function parameterized by a weight vector obtained from an exponentially moving average of the Q-function weights to stabilize training. The policy improvement step then attempts to project a parametric policy $\pi(\mathbf{a}_t|\mathbf{s}_t)$ by minimizing KL divergence between the policy and a Boltzmann distribution induced by the Q-function using the following objective

$$J(\pi) = \mathbb{E}_{\mathbf{s}_t\sim\mathcal{D}}\left[D_{\mathrm{KL}}(\pi(\cdot|\mathbf{s}_t)||\mathcal{Q}(\mathbf{s}_t,\cdot))\right]$$

where $\mathcal{Q}(\mathbf{s}_t,\cdot) \propto \exp\{\frac{1}{\alpha}Q(\mathbf{s}_t,\cdot)\}$.

For modelling the reconstruction loss, the authors then adopt the RAE approach of [10] which imposes a L_2 penalty on the learned representation $\mathbf{z}_t$ and weight-decay on the decoder parameters.

$$J(\mathrm{RAE}) = \mathbb{E}_{\mathbf{o}_t\sim\mathcal{D}}\left[\log p_\theta(\mathbf{o}_t|\mathbf{z}_t) + \lambda_{\mathbf{z}}||\mathbf{z}_t||^2 + \lambda_\theta||\theta||^2\right]$$

They also prevented the actor's gradients from updating the convolutional encoder. Unfortunately, this slowed down signal propagation to the encoder, and thus they found it important to update the convolutional weights of the target Q-function faster than the rest of the network's parameters. They therefore employed different rates τ_Q and τ_{enc} (with $\tau_{\mathrm{enc}} > \tau_Q$) to compute the Polyak averaging over the corresponding parameters of the target Q-function.

2.2. DRQ

In [11], the authors propose the use of image augmentation for model-free reinforcement learning. They suggest that use of image augmentation enables more robust learning. However their method did not include any auxiliary losses or pre-training. They used common image augmentation techniques used in computer vision tasks, and observed that it resulted in significant boost in SAC's performance on DeepMind control suite. Their approach of using image augmentation is very flexible and simple to include in any model-free algorithm. They also demonstrated the simplicity by applying their method with DQN[9] and improved performance in terms of total training required to reach competitive results on Atari 100k [12] benchmark.

2.3. DRQ:v2

DRQ:v2 was an extension to DRQ [11]. They identified bottlenecks in their approach such as as replay buffer management, data augmentation processing, batch size, frequency of learning updates, and improved upon them. They used DDPG algorithm[8] for demonstration of their technique, added possibility of using multi-step return, added bilinear interpolation, introduced an exploration schedule and improved hyperparameters.

3. APPROACH

3.1. Reproducing SAC+AE

We test the reproducibility of the model proposed in [5], as we would later submit our findings to **ML Reproducibility Challenge 2021**. We consider the same model structure and value of hyperparameters as considered in the original paper [5]. An overview of all the hyper parameters is depicted in Table 4 of [5].

3.2. $\mathcal{AD}$-model

We now present our approach for the $\mathcal{AD}$-model, which is based on using image reconstruction loss as in [5] and using image augmentation as in [7].

The basic setup is shown in Figure 1, where we have the following components - 1) Encoder, 2) Decoder, 3) Augmentation unit, 4) Actor, 5) Critic. The Actor and Critic are ac-

Algorithm 1 $\mathcal{AD}$(1,1)-Algorithm.

```
Inputs:
$f_\xi$, $\pi_\phi$, $Q_{\theta_1}$, $Q_{\theta_2}$: parametric networks for encoder, policy, and Q-functions respectively.
aug: random shifts image augmentation.
$\sigma(t)$: scheduled standard deviation for the exploration noise
$T$, $B$, $\alpha$, $\tau$, $c$: training steps, mini-batch size, learning rate, target update rate, clip value.
Training routine:
for each timestep $t = 1..T$ do
    $\sigma_t \leftarrow \sigma(t)$                                            ▷ Compute stddev for the exploration noise
    $\boldsymbol{a}_t \leftarrow \pi_\phi(f_\xi(\boldsymbol{x}_t)) + \epsilon$ and $\epsilon \sim \mathcal{N}(0, \sigma_t^2)$      ▷ Add noise to the deterministic action
    $\boldsymbol{x}_{t+1} \sim P(\cdot|\boldsymbol{x}_t, \boldsymbol{a}_t)$                               ▷ Run transition function for one step
    $\mathcal{D} \leftarrow \mathcal{D} \cup (\boldsymbol{x}_t, \boldsymbol{a}_t, R(\boldsymbol{x}_t, \boldsymbol{a}_t), \boldsymbol{x}_{t+1})$         ▷ Add a transition to the replay buffer
    UPDATECRITIC($\mathcal{D}, \sigma_t$)
    UPDATEACTOR($\mathcal{D}, \sigma_t$)
    UPDATEDECODER($\mathcal{D}, \sigma_t$)
end for
procedure UPDATECRITIC($\mathcal{D}, \sigma$)
    $\{(\boldsymbol{x}_t, \boldsymbol{a}_t, r_{t:t+n-1}, \boldsymbol{x}_{t+n})\} \sim \mathcal{D}$                     ▷ Sample a mini batch of $B$ transitions
    $\boldsymbol{h}_t, \boldsymbol{h}_{t+n} \leftarrow f_\xi(\mathrm{aug}(\boldsymbol{x}_t)), f_\xi(\mathrm{aug}(\boldsymbol{x}_{t+n}))$        ▷ Apply data augmentation and encode
    $\boldsymbol{a}_{t+n} \leftarrow \pi_\phi(\boldsymbol{h}_{t+n}) + \epsilon$ and $\epsilon \sim \mathrm{clip}(\mathcal{N}(0, \sigma^2))$      ▷ Sample action
    Compute $\mathcal{L}_{\theta_1,\xi}$ and $\mathcal{L}_{\theta_2,\xi}$                              ▷ Compute critic losses
    $\xi \leftarrow \xi - \alpha \nabla_\xi (\mathcal{L}_{\theta_1,\xi} + \mathcal{L}_{\theta_2,\xi})$                  ▷ Update encoder weights
    $\theta_k \leftarrow \theta_k - \alpha \nabla_{\theta_k} \mathcal{L}_{\theta_k,\xi} \quad \forall k \in \{1, 2\}$              ▷ Update critic weights
    $\bar{\theta}_k \leftarrow (1 - \tau)\bar{\theta}_k + \tau \theta_k \quad \forall k \in \{1, 2\}$              ▷ Update critic target weights
end procedure
procedure UPDATEACTOR($\mathcal{D}, \sigma$)
    $\{(\boldsymbol{x}_t)\} \sim \mathcal{D}$                                   ▷ Sample a mini batch of $B$ observations
    $\boldsymbol{h}_t \leftarrow f_\xi(\mathrm{aug}(\boldsymbol{x}_t))$                              ▷ Apply data augmentation and encode
    $\boldsymbol{a}_t \leftarrow \pi_\phi(\boldsymbol{h}_t) + \epsilon$ and $\epsilon \sim \mathrm{clip}(\mathcal{N}(0, \sigma^2))$          ▷ Sample action
    Compute $\mathcal{L}_\phi$                                         ▷ Compute actor loss
    $\phi \leftarrow \phi - \alpha \nabla_\phi \mathcal{L}_\phi$                               ▷ Update actor's weights only
end procedure
procedure UPDATEDECODER($\mathcal{D}, \sigma$)
    $\{(\boldsymbol{x}_t)\} \sim \mathcal{D}$                                   ▷ Sample a mini batch of $B$ observations
    $\boldsymbol{h}_t^{enc} \leftarrow f_\xi((\boldsymbol{x}_t))$                                ▷ Apply the encoder on data
    $\boldsymbol{x}_t^{dec} \leftarrow f_\xi^{-1}((\boldsymbol{h}_t^{enc}))$                           ▷ Apply the decoder on encoded data
    Compute $\mathcal{L}_{latent}$ by averaging over all pixels of $\boldsymbol{h}_t^{enc}$          ▷ Compute latent loss
    Compute $\mathcal{L}_{recon} \leftarrow \mathrm{MSE}(\boldsymbol{h}_t^{enc}, \boldsymbol{x}_t^{dec})$    ▷ Compute image reconstruction loss by mean squared error between reconstructed and original image
    Compute $\mathcal{L}_{AE} \leftarrow \mathcal{L}_{recon} + \lambda \cdot \mathcal{L}_{latent}$              ▷ Compute total reconstruction loss
    $\xi \leftarrow \xi - \alpha \nabla_\xi \mathcal{L}_{AE}$    for both decoder and encoder params    ▷ Update actor encoder and decoder weights
end procedure
```

cording to the DDPG Framework. In our setup we can choose to either keep or remove the augmentation and decoder, which gives us four variations which we refer as $\mathcal{AD}$(0,0), $\mathcal{AD}$(0,1), $\mathcal{AD}$(1,0) and $\mathcal{AD}$(1,1) to indicate whether these components are included in model or not. We now describe the structure of the components in detail.

Image Augmentation: We use random shifts for image augmentation. Precisely, the 84x84 images are first padded by 4 pixels (by repeating the boundary pixels), then a random 8484 section of it is picked, which effectively is similar to a random shift of the original image. As in [7], bilinear interpolation is also done (where each pixel is replaced by average of the four neighbouring pixels).

Image Encoder: The image encoder is used to embed the observation (the image) into a low-dimensional latent representation using convolution layers. The encoder consists of four convolutional layers with 32 filters and kernel size of 3x3. Each of these layers is followed by ReLU activation function. The first layer has stride of 2, while the other layers have stride of 1. This structure is same as used in [7].

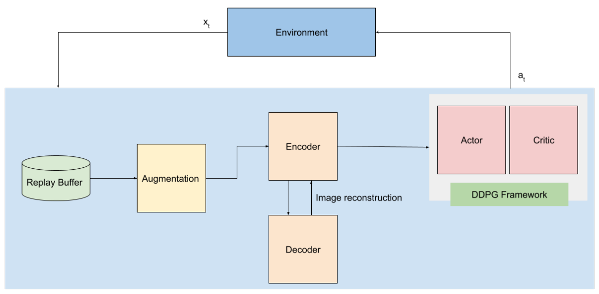


Fig. 1. Overview of $\mathcal{AD}$-model

Image Decoder: The image decoder network is symmetric to encoder except that ConvTranspose2d layers are used.

Actor: The actor network consists of two components, trunk and policy network. The trunk is a simple network with a dense layer followed by normalization after which tanh activation function is applied. The trunk network transforms the input latent vectors to vector which we call as feature vector, which is input to the policy network. Following DDPG principles, the policy network outputs a vector of dimension same as that of action space. We call this vector as the mean vector, and use a fixed value as standard deviation to get a multivariate gaussian distribution over the action space which gives us a stochastic policy. The policy network consists of a single hidden layer with dimension of 1024.

Critic: The critic network has a trunk and Q networks. The structure and purpose of the trunk is same as that for actor, however the trunk is different for both in the sense that this component is not shared between Actor and Critic. The critic consists of a Q-network and a target network, where periodically the weights of target network are updated by Polyak averaging. The Q-network take input of size of feature vector + size of action space as input, and output a single value which is the approximation for the q value for the given state and action. The loss functions involved are:

$$\mathcal{L}_{\theta_k,\xi}(\mathcal{D}) = \mathbb{E}_{\tau\sim\mathcal{D}}\big[(Q_{\theta_k}(\boldsymbol{h}_t,\boldsymbol{a}_t) - y)^2\big] \quad \forall k \in \{1,2\}, \quad (1)$$

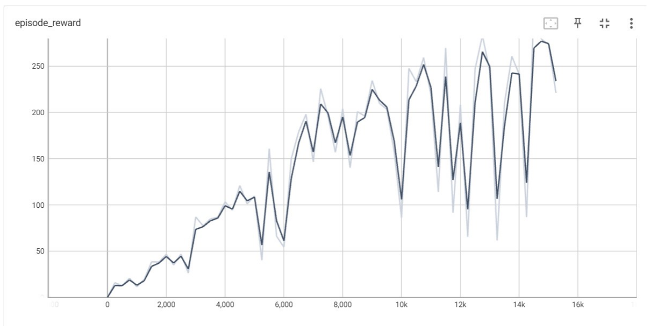
$$\mathcal{L}_{\phi}(\mathcal{D}) = -\mathbb{E}_{\boldsymbol{x}_t\sim\mathcal{D}}\big[\min_{k=1,2} Q_{\theta_k}(\boldsymbol{h}_t,\boldsymbol{a}_t)\big], \quad (2)$$

$$\mathcal{L}_{AE}(\mathcal{D}) = \mathbb{E}_{\rtimes_t\sim\mathcal{D}}\big[\mathrm{MSE}(\mathbf{o}_t,\mathbf{z}_t) + \lambda_{\mathbf{z}}||\mathbf{z}_t||^2\big] \quad (3)$$

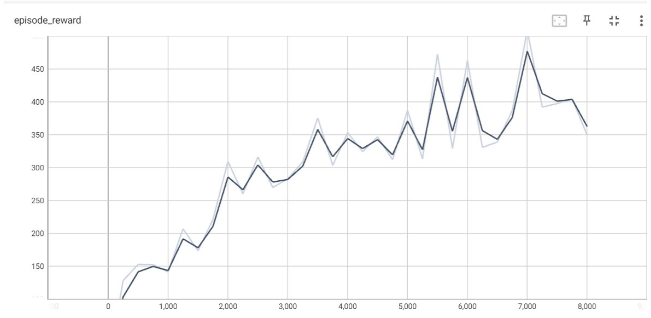
4. RESULTS

As clear from previous sections, we perform two different tasks (1) Reproduce SAC+AE [5], and (2) Experiment with our implementation of combination of Image Augmentation and Image Reconstruction Loss in DDPG. Our observations, results and possible explanations are mentioned in the next two subsections.

4.1. Reproduce SAC+AE



(a) SAC+AE Cheetah Run



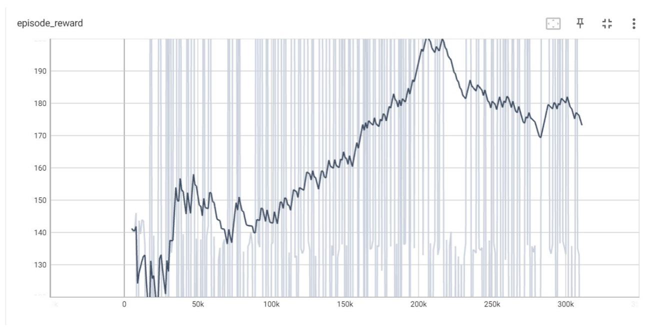
(b) SAC+AE Walker Walk

Fig. 2. Reproduced results from SAC+AE on two different tasks.

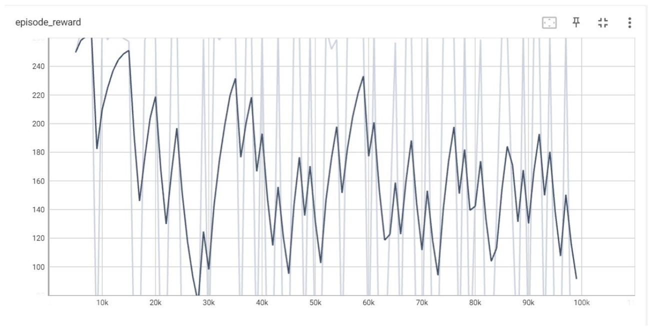
We begin with [5] as a base and reproduce the SAC+AE technique implemented in the paper. As a part of results for reproducibility challenge, we provide the following conclusions:

- The first command to create the environment is buggy and requires hefty amounts of debugging. Instead we manually installed all the versions of dependencies.
- Logic conveyed in the paper is aptly implemented in the code as well. (We provide a schema of steps to follow to manually setup the environment in our code)
- With our current experimental setup on GCP with 16 cores, 104 GB RAM and 1xTesla T4 GPU, the experiment took more than 12 hours to simulate 16,000 training steps. So it is hard to compare with the original results of the paper, since their results were over 10^6 order of training steps.

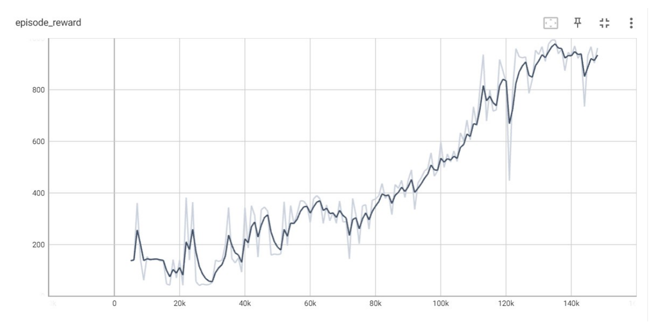
The reproduced results for the two consider tasks of (1) Cheetah Run, and (2) Walker Stand are as depicted in Figure 2.



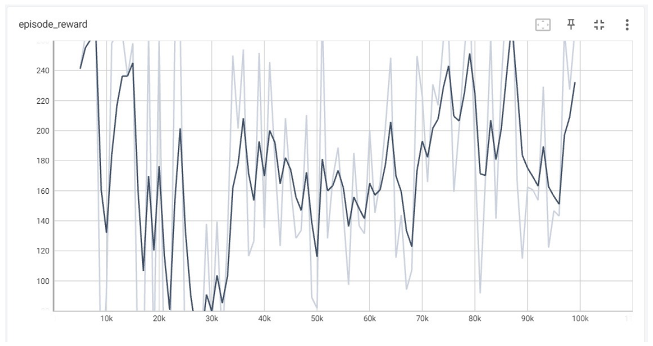
(a) No Image Aug or Recon Loss



(b) Recon Loss



(c) Image Aug



(d) Image Aug + Recon Loss

Fig. 3. Our Experimental results of different combinations of Image Augmentation and Reconstruction Loss techniques. We use an Easy Task of Walker Stand as a benchmark to compare all the models.

4.2. $\mathcal{AD}$-model

Now we analyze our implementation of four $\mathcal{AD}$-models on $Walker\_Stand$ task for 99,000 training steps. Evaluation is done over 1000 episodes after ever 10,000 training steps.

Figure 3 depicts the episode rewards over our four variations of $\mathcal{AD}$-model. Also we can observe the final episode rewards at the end of 97,000 training steps, for both evaluation as well as training in Table 1.
We draw the following conclusions based on our experimental outputs from Figure 3 and Table 1:

- Out of all four models, the $\mathcal{AD}$(1,0) model (Figure 3(c)) outperforms in the $Walker\_Stand$ task. From Table 1 as well, it clear that by the end of our training, maximum reward corresponds to $\mathcal{AD}$(1,0).

- In $\mathcal{AD}$(0,1) (Figure 3(b)) i.e. the model with reconstruction loss and a decoder added to the DDPG, the results are negative, as the average episode reward tends to decrease with time.

- From Figure 3(a), we can see the average expected reward growing with time for $\mathcal{AD}$(0,0) i.e. the model with no image augmentation or reconstruction. However, the highest value (optimal) is not attained in the considered training time.

- For $\mathcal{AD}$(1,1) (Figure 3(d)), i.e. the model with both image augmentation and image reconstruction loss on DDPG, the values of average episode reward while training, tends to oscillate with great variation. However, by the end a small peak in the values is observable.

One possible reason for the aforementioned observations might be that image augmentation and reconstruction loss are although good strategies individually, they have conflicting effects on the encoder. Use of image augmentation would encourage learning similar latent vectors for augmented images, while if the latent representation for the normal and augmented images are similar, then image reconstruction loss will suffer as the decoder will not decode the latent vector correctly as the original and augmented image have similar latent vector.

Another possible for the aforementioned observations might be that the model considered is Easy and using complex models such as $\mathcal{AD}$(1,1) i.e. using both image augmentation and a decoder incorporating image reconstruction loss may turn out to be too complex for an easy task like $walker\_stand$.

Finally, our experiments were limited to 99k frame steps, therefore some variation may be better but may require more training.

Model	**Episode Reward**	
	Eval	Train
$\mathcal{AD}$(0,0)	241.2235953	363.5182619
$\mathcal{AD}$(0,1)	155.6814344	266.1268151
$\mathcal{AD}$(1,0)	503.7203478	555.4227242
$\mathcal{AD}$(1,1)	269.6645507	265.9363906

Table 1. A comparison of Episode Reward Values on evaluation and training at the end of 97k training steps. The evaluation is done on 1000 episodes after every 10k training steps.

5. CONCLUSION AND FUTURE DIRECTIONS

From our experiments, we observe that the combining the techniques of image augmentation and auxiliary losses like image reconstruction loss does not perform good with Easy tasks such as $walker\_stand$. However, because of system restrictions, these observations are totally based on observations from seemingly less number of training steps (10^5) compared to 10^6 steps for which both the base paper [11], [7] ran their experiments.

A major question that using DRQ (Image Augmentation) piques is about "How can we learn good representations from unlabeled data?". In recent years, we have seen an explosion of unsupervised Deep Learning methods based on these principles. In fact, some self-supervised contrastive-based representations already match supervised-based features in linear classification benchmarks. So another future replacement of DRQ might be Contrastive Unsupervised learning (CURL [13]) and is worthy to explore. One more approach can be trying for a combination of image augmentation techniques on contrary to [6] which uses only $RandomShift$. We can also test the robustness of these models by introducing some background noise in the images as well.

References

[1] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[2] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al., "Soft actor-critic algorithms and applications," *arXiv preprint arXiv:1812.05905*, 2018.

[3] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu, "Reinforcement learning with unsupervised auxiliary tasks," *arXiv preprint arXiv:1611.05397*, 2016.

[4] Debidatta Dwibedi, Jonathan Tompson, Corey Lynch, and Pierre Sermanet, "Learning actionable representations from visual observations," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 1577–1584.

[5] Denis Yarats, Amy Zhang, Ilya Kostrikov, Brandon Amos, Joelle Pineau, and Rob Fergus, "Improving sample efficiency in model-free reinforcement learning from images," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 12, pp. 10674–10681, May 2021.

[6] Denis Yarats, Ilya Kostrikov, and Rob Fergus, "Image augmentation is all you need: Regularizing deep reinforcement learning from pixels," in *International Conference on Learning Representations*, 2021.

[7] Denis Yarats, Rob Fergus, Alessandro Lazaric, and Lerrel Pinto, "Mastering visual continuous control: Improved data-augmented reinforcement learning," *arXiv preprint arXiv:2107.09645*, 2021.

[8] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra, "Continuous control with deep reinforcement learning," 2019.

[9] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[10] Partha Ghosh, Mehdi SM Sajjadi, Antonio Vergari, Michael Black, and Bernhard Schölkopf, "From variational to deterministic autoencoders," *arXiv preprint arXiv:1903.12436*, 2019.

[11] Ilya Kostrikov, Denis Yarats, and Rob Fergus, "Image augmentation is all you need: Regularizing deep reinforcement learning from pixels," 2020.

[12] Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, et al., "Model-based reinforcement learning for atari," *arXiv preprint arXiv:1903.00374*, 2019.

[13] Aravind Srinivas, Michael Laskin, and Pieter Abbeel, "CURL: contrastive unsupervised representations for reinforcement learning," *CoRR*, vol. abs/2004.04136, 2020.