Supervised Object State Prediction to Improve Sample-Inefficient Pixel-based Policies

Aditya Arjun

***Abstract*—Empirically, it is much more difficult to perform reinforcement learning from high-dimensional pixel observations compared to reinforcement learning from low dimensional object state observations, in terms of sample efficiency and stability. Intuitively, this is because the model needs to extract a meaningful, compressed representation of noisy high-dimensional image data before learning can proceed. In the real world, low dimensional object state observations may not always be available (as position and orientation data for objects that a robot might need to interact with), but this data is available in robotics simulation environments used to train reinforcement learning policies. In this paper, we explore leveraging low dimensional object state observations in simulation to train reinforcement learning policies that can operate on pixel observations when deployed. The low dimensional object state available in simulation can be used as a supervised objective for a separate module that aims to predict object state from pixel observations, and a reinforcement learning policy that is trained on object state can use this module's predicted object state to take actions at test time. We analyze this approach using standard tasks in robosuite, a framework for robotics reinforcement learning [12]. In general, this proposed approach leads to policies that follow reasonable trajectories and come close to solving difficult robotics manipulation tasks, but compounding small-errors in object state prediction tend to lead to covariate shift issues that stop these models from fully solving manipulation tasks**

I. Introduction

Generally, reinforcement learning from pixel states is more sample-inefficient than learning from state based features [11]. The information necessary to predict state is often in raw image observations, but machine learning models have difficulty learning and extracting this state information from high dimensional image observations. There are a variety of approaches that other literature takes to learn a robust representation of image data that is easier and more stable to learn from than the raw image data. For example, some works use image augmentations to learn a representation invariant to the augmentations [8, 11] and others use auxiliary loss terms to force the encoded state created from the image to help predict useful semantic information [6, 7].

Within the robotics domain specifically, there are many tasks where state observations may be available in simulation, but are not available in the real world, as lifting a block or moving a block to a goal location [12]. It is desirable to produce a pixel-based policy that can be used in the real world, and the simulation environment in this special case provides extra information that can be used to inform policy learning in learning a pixel-based policy. Previous works that produce pixel-based policies do not leverage the existence of this potential new source of information during training likely because this difference in available information in simulation and the real world is not necessarily present in general case reinforcement learning domains.

Rather than using a purely unsupervised or self-supervised approach with the pixel observations in simulations, we can leverage the object state as a supervised objective to predict and train off the raw object state. When deployed, rather than using the object state (which is no longer available) the model trained using supervised learning to predict object state is used to estimate the object state, producing a policy that runs off of pixel observations, but was effectively trained with state observations. Ideally, this trained policy will not suffer from the sample-inefficiency of policies trained on pixels using previous approaches and still achieves a similar level of performance to these policies. The observations used to train the supervised component will be built up over time as the agent dynamically collects new observations with experience. This approach can leverage many of the data-augmentation techniques discussed in related works in the supervised portion of the algorithm, and can be seamlessly integrated with many existing RL algorithms as the supervised portion of the model can be trained almost independently of the RL model, as the soft actor critic implementation used in this paper. In this paper, we explore this approach of using a supervised objective to predict the object state from images to help solve robotics manipulation tasks. By using this supervised objective to guide representation of the image state, we hypothesize that trained RL policies operating on pixels will be more stable and sample-efficient to train.

While policies trained using the proposed approach tend to follow similar trajectories to policies trained exclusively off of ground truth object state, small errors in object state prediction can compound and lead the policy to new states and result in task failure.

A. Problem Statement

We are given two Markov decision processes for a robot learning task (MDP), one for the training environment and one for the test time environment, defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R})$, where $\mathcal{S}$ denotes the set of states, $\mathcal{A}$ denotes the set of actions, $\mathcal{T}$ gives transition probabilities where $\mathcal{T}(s', a, s) = P(s_{t+1} = s' | s_t = s, a_t = a)$, and $\mathcal{R}$ gives the reward where $\mathcal{R}(s, a)$ is the reward given to the agent for being in state s and executing action a. In the training environment, $\mathcal{S}_{train}$ consists of both the pixel observation and object state observation, but in the test environment $\mathcal{S}_{test}$ only consists of

the pixel observation. Otherwise, the training and test MDPs are identical.

The goal is to leverage the extra information available in the training MDP to train a policy that optimizes for reward on the test MDP. Specifically, the goal is to train a policy π that maximizes the expected rewards over a trajectory in the test time MDP

$$\sum_t \mathbb{E}_{s_t \in \mathcal{S}_{\text{test}}, a_t \sim \pi} R(s_t, a_t)$$

II. Related Work

Other works that attempt to reduce the sample-efficiency gap between learning from state and learning from pixels do not take advantage of object state and solely train based on pixel state. Because these approaches do not use the raw object state at all, for these approaches there is no functional difference between the training and testing MDP, and they simply seek to optimize and learn a good unsupervised or self-supervised representation for image data that is more stable than just the raw pixels themselves.

A. Autoencoders and Auxiliary Loss Terms

Lange and Riedmiller present an approach that uses deep autoencoders to create an encoded state that ideally contains useful semantic information important for solving the task [7]. By reducing the dimensionality of images to the encoded state, training becomes much more stable. However, the actual RL task and rewards have no influence on the encoding itself, so the encoding may not contain useful task information and the resulting policy may be suboptimal.

Jaderberg et al. use auxiliary losses based on pixel changes and network features to help the agent learn a good representation of images that will aid learning [6]. However, the auxiliary loss terms may not be directly useful to all tasks and may lead to suboptimal policies if the agent attempts to optimize one of the auxiliary losses over the primary objective.

This family of approaches learn good representations of image state, but the actual object state is often a superior encoded state that provides a more useful learning signal for the problem. In this specific RL instance where the object state is available during training, it makes more sense to directly use that rather than learn an unsupervised representation.

B. Data Augmentation

Laskin et al. investigate adding various data augmentations as cropping, translating, and color jitter to sampled image observations [8]. This data augmentation is a similar idea used in computer vision supervised learning, which forces the model to learn useful features about the image state that are invariant to small changes that should not affect the meaning of the image. While RAD did demonstrate success on simple domains, they do not demonstrate success on more complicated robotics tasks or in robotics simulation frameworks.

Srinivas et al. investigate the same idea as [8], but train a constrastive loss objective rather than directly augmenting image observations [11].

Fan et al. presents a more recent approach that aims to leverage data augmentation effectively by training an expert using simple, weak augmentations and then training a student that uses stronger data augmentations and supervises the student on the stronger augmented data to predict what the expert predicted [2]. Secant does demonstrate success on Robosuite, a robotics simulation framework that provides useful tasks to benchmark RL policies on [12].

These approaches are useful because they generate good representations of images that are invariant to small, inconsequential changes. However, given that the problem can be solved by using solely object state, using the object state to train the part of the model that processes images seems likely to provide a stronger signal than the only the reinforcement learning reward back-propagated through the full model pipeline, and the supervised component of our proposed approach can similarly leverage data augmentations that these alternative approaches use in training as the supervised component is initially independent of the RL agent training off of object state.

III. Data

For this project, we used robosuite, a simulation framework for robot learning to train and evaluate reinforcement learning policies [12]. We ended up testing this approach on two single robot tasks (Reach and Lift) from robosuite, using the OSC_POSE controller, the Panda robot, and a control frequency of 20 Hz. The environment episodes were run with similar parameters to robosuite benchmark, using a horizon of 500 time steps for each task. Image observations and object state observations for both environments were stored and used to train the supervised object predictor and RL Agent. The image observations consisted of a 96x96 image from the agentview camera observation, and the object state observations differed based on the task as described below. The policy also had access to the robot's state, which included information as the position and orientation of the robot's gripper in both the train time and test time environment. It makes sense to give the policy access to the robot state as real-world robots will have access to robot sensor data during deployment.

A. Reach Task

The reach task is a simplified custom task designed to test the robot's ability to reach a given fixed target location. The policy receives a shaped reward based on Euclidean distance to the goal location based on the current location of the gripper.

There are technically no objects in this simplified reach task, so we ended up designing our own object representation for the task that would serve as a sufficient information to solve the task. The low-dimensional object state used as a supervised objective was simply the difference between the robot's gripper position and the desired goal position. This serves as a useful object state for the task as the robot only needs to learn to minimize the difference between the robot's gripper and the goal position to solve the task.

B. Lift Task

The lift task is a standard task that is already built into robosuite desired to test the robot's ability to navigate to a target cube object and lift it up. The policy receives a shaped reward based on its distance to the target object, whether the arm is gripping the target object, and if the object is successfully being lifted.

The object state in the Lift task consists of the position of the cube object, orientation of the cube object (as a quaternion), and the distance between the robot gripper and the cube. Like in the reach task, the robot needs to learn to minimize the difference between the gripper and the cube using the object state and then use the orientation and position information to successfully grip the cube.

IV. Methods

We ended up exploring two primary approaches: an end-to-end approach that trains all main components together and an "imitation learning" approach that trains the main components separately by using a trained reinforcement learning agent to generate expert rollouts for an object predictor to train from. Below we describe the main components used in both approaches, explain how the main components are deployed at test time, detail both approaches more explicitly, and compare the two approaches.

A. Main Components

1) Object Predictor: The object predictor is a relatively straightforward CNN architecture, as convolutional neural networks have been quite successful in a variety of supervised learning computer vision tasks [3]. This CNN architecture consists of three 2D Conv Layers, followed by a flatten operation into two feedforward layers. The object predictor will be trained based on the images of the rollout taken by the RL agent and ground truth object states. Because the object predictor is fully supervised, we can leverage standard computer vision techniques for training, as the data augmentation discussed in the literature review.

2) RL Agent: The RL agent simply needs to learn find actions using a low-dimensional state observation encompassing important information about the environment to optimize reward, and existing standard reinforcement learning techniques exist to solve this common problem. Robosuite has been benchmarked with standard reinforcement learning approaches that are able to solve this problem on some of the built in tasks on robosuite. The RL agent will simply be an implementation of Soft Actor Critic, a well-known off-policy reinforcement learning algorithm [4].

B. Test Time Pipeline

Given an object predictor that can predict object state from images and RL agent operating off of object state, we can construct a simple pipeline to create a policy that is able to operate off of image observations by connecting the two main components. Given some image, we can run the object predictor to predict the object state and pass this predicted object state to the RL agent to determine some action. Given that the object predictor is able to accurately predict the object state from the image, the RL agent can simply make decisions based on object state, which is an easier, solved task in robosuite.

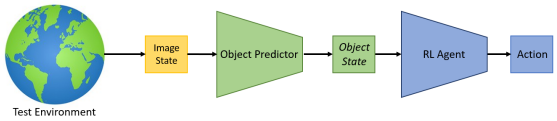


Fig. 1: Depiction of Pipeline during test time. The image state is given to the Object Predictor, which gives a predicted object state that the RL agent operates off of.

C. End to End Approach

Initially, the RL agent will train purely off the true object state returned from the environment. As the RL agent visits more states, these states will be saved in a replay buffer and the object predictor will periodically sample a large batch from the replay buffer and train to predict the object state of each image in that batch from the raw pixels. After some number of pretraining iterations, specified as a hyperparameter, the RL agent will begin to use a linear combination of the true object state and the object predictor's predicted state. If the true object state is $\mathbf{o}$ and the object predictor's predicted state is $\tilde{\mathbf{o}}$, then the RL agent will take $(1-\alpha)\cdot\mathbf{o}+\alpha\cdot\tilde{\mathbf{o}}$ as input, where $0\leq\alpha\leq 1$ and α begins at 0 at the start of the run. Each iteration, α linearly increases by some amount, until reaching the cap of 1, at which point the RL Agent is completely using the estimated object state and can be deployed once it is sufficiently trained and able to accomplish the task using the estimated object state.

After this process is finished, the result can be plugged into the test time pipeline, giving a reinforcement learning policy that operates on images but was trained as if it operated off states that ideally will demonstrate improved sample-efficiency compared to standard image-based reinforcement learning approaches.

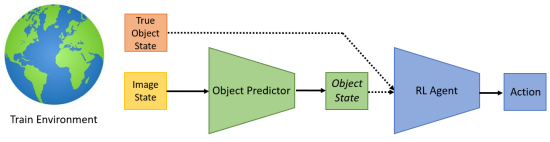


Fig. 2: Depiction of end to end training scheme. The RL Agent is trained with a linear combination of the ground truth object state and the predicted object state given by the Object Predictor.

D. "Imitation Learning" Approach

This approach is not true imitation learning, as neither of the components are being trained to optimize a policy based on expert demonstrations, but does share similarities to behavioral

cloning, a common imitation learning technique. Rather than learning to copy an expert's policy in each state as in standard behavioural cloning, this approach uses a separately trained reinforcement learning policy off of object state to generate several rollouts of expert data. Then, rather than predicting the action along each of this policy's trajectories, the object predictor is trained to predict the object state that the policy uses given the image observation at each state along the trajectory. The RL agent trained in isolation off of the ground truth object state serves as the expert and the object predictor learns to predict object state along the trajectories that the trained RL agent tends to follow.

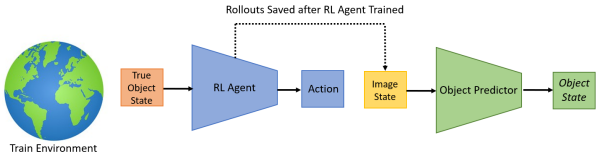


Fig. 3: Depiction of imitation learning training scheme. The RL agent is purely trained off of ground truth object state and used to generate several expert data rollouts once it is able to solve the desired task. The object predictor is then trained independently on these data rollouts to predict the object state from images along trajectories that the RL agent follows.

E. Comparison of Approaches

The "Imitation Learning" approach initially was developed to present a less memory-hungry and stable approach to developing the desired pipeline compared to the end to end approach. In the end to end approach, which uses Soft Actor Critic as the RL Agent, we maintain a replay buffer of previously seen observations to train the policy and critic offline. Because the end to end approach trains the object predictor and the RL agent together, this replay buffer needs to maintain a massive collection of high dimensional image observations, which is an expensive memory cost. Furthermore, if the supervised predictor happened to be unstable initially in training, it might cause instability and exploding losses in the actor and critic networks as these networks would be receiving noisy inputs and making bad predictions. As the imitation learning approach trains these models in isolation, this noise issue does not occur and there is no need to store high dimensional image observations in the replay buffer in-memory (as they can simply be saved to disk after the RL agent is trained).

However, the "Imitation Learning" approach may suffer from common issues in behavioural cloning as those brought up in [10]. The object predictor is trained in isolation from the RL agent and only along trajectories that the RL agent encounters. If the object predictor has small errors in the predicted object state passed to the policy, this noise might cause the policy to make different actions and change the state to a new area that the object predictor does not accurately deal with. This issue may cause these small errors to compound and cause the overall policy to fail the task. The end to end learner uses the object predictor while generating rollouts, so the policy is trained to be more robust to the noise of the object predictor model and therefore will likely not suffer from the same issues.

We did end up fixing a memory issue that Dr. Zhu brought up during the paper spotlight that limited the size of the replay buffer by storing the high dimensional image observations as bytes rather than casting the images to floats, which saved significant memory and made the end to end approach more practical.

F. Base Code

In this study, we built off and modified the robosuite benchmark code [5] to run experiments and handle logging and used a Reinforcement library named rlkit [1] to handle the base soft actor critic implementation and infrastructure needed for soft actor critic.

V. Experiments

For these experiments, the policies were trained for 500 epochs according to each approach, where each epoch consisted of 2500 exploration steps and 1000 training steps. Afterwards, the final policies were evaluated over 20 different rollouts to determine the mean reward achieved along the rollouts, the maximum number of rollouts, and the percentage of rollouts where the policy successfully finished the task.

A. End to End Approach Results

Unfortunately the end to end policies were not able to solve either of the two tasks, reach or lift in the desired number of epochs. However, learning did appear to be proceeding for both tasks and both tasks were able to receive significantly better rewards than policies acting at random. For the reach task, the end to end approach was able to learn to occasionally move close to the desired location, evidenced by the high max reward close to solving the task. For the lift task, the end to end approach was able to navigate towards the cube and claim the associated reward, but was not able to successfully grasp the cube. We hypothesize that this is due to small errors in object state prediction causing the model to operate as if it was in an incorrect state and attempt to grasp the cube object when the cube object is far outside the robot gripper. Given that a policy trained purely off of object state is able to solve both tasks within 500 epochs, as demonstrated by [5], the end to end approach and errors in object state prediction make it harder for the overall policy to learn and decreases sample efficiency some.

Task	Mean Reward	Max Reward	Success Percentage
Reach	0.0414	0.8277	0%
Lift	0.113	0.418	0%

Fig. 4: Table compiling results for end to end approach results

B. "Imitation Learning" Approach Results

The imitation learning approach did significantly better than the end to end approach on the evaluation and was able to solve both tasks at least once out of 20 rollouts. The imitation learning did significantly better on the reach task, which is understandable as this task requires less precision than the lift task, which needs good positioning of the robot gripper to accurately pick up the object. Additionally, it's possible to solve the reach task using only robot state observations, which are given independent of the object predictor, so it is possible the model is simpler learning to optimize based on the robot sensor state observations and does not need to deal with the noisy object state predictions in the reach case. In the Lift case, the imitation learning approach suffers from compounding noise issues that cause the gripper to badly estimate the object state and attempt to grasp when out of position.

Task	Mean Reward	Max Reward	Success Percentage
Reach	0.867	1.0	95%
Lift	0.178	1.0	5%

Fig. 5: Table compiling results for imitation learning approach results

C. Qualitative Analysis of Common Issues

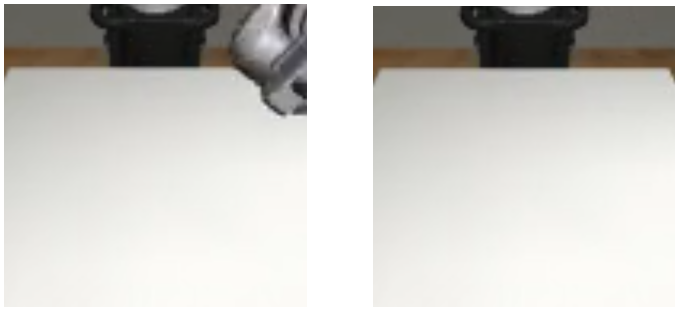
In this section, we analyze common issues encountered while implementing the supervised object state predictor and how these common issues might contribute to task failure.

1) Object Occlusion: During rollouts where the robot needs to interact with a physical object, the robot might occlude the view of the object that it needs to interact with, which practically makes it very difficult to predict the occluded object's position and orientation. The below figure demonstrates this observed issue while the robot was attempting to perform the Lift task. Given that the object state cannot be reasonably inferred from the pixel observation while the object is occluded, the policy is likely operating off of bad object state predictions in these cases, which leads to unpredictable behaviour and task failure.

(a) Arm moving over object (b) Arm occluding object

Fig. 6: Demonstration of the object occlusion issue. In the Lift task, the robot arm can cover the object and make it difficult to predict position and orientation.

2) Robot Out of View: The robot arm can even leave the field of view of the agent view camera during rollouts, which was often observed in the Reach task when the arm overshot the target position. Given that the arm is outside the field of view, it is difficult to infer elements of the object state that rely on the relative position of the robot arm and the target position or object, as the robot arm can be in a variety of positions outside of the field of view that result in a different relative position but an identical pixel observation. While it might technically be possible to learn to deal with this issue by using the robot state observations, the object predictor currently only uses the raw image state to make object state predictions and therefore will have difficulty predicting object state in these cases.



(a) Arm moving towards target (b) Arm overshooting target and leaving view

Fig. 7: Demonstration of the out of view issue. In the Reach task, the robot arm can leave the field of view, making it difficult to accurately predict object state from just the image observation.

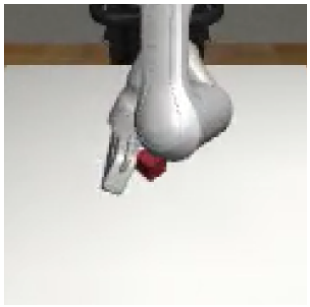
3) Object Prediction Error: This approach of predicting object state can suffer from small errors in prediction of the object state. Small inaccuracies in the object predictor can cause the policy to act as if it were in a different position and fail the task, especially in manipulation tasks where somewhat precise grasping is needed. This issue is especially evident in the imitation learning implementation, as these errors can bring the object predictor to different observation distributions outside of the object predictor training distribution, which causes further compounding errors.

D. Comparison of Approaches to Object State Policies

Unfortunately, the end to end approach does suffer from some sample inefficiency issues compared to object state prediction, likely because of the increased difficulty in learning from noisy object state observations. Figure 9 displays this slight sample inefficiency gap, as the state based policies are able to solve the task much quicker than the end to end approach.

By virtue of the imitation learning approach only using the policy trained off of object state to generate image observations, the imitation learning approach is much more sample efficient than the end to end approach. While the current

(a) State-based policy grasps object successfully



(b) Same policy with object predictor misses object slightly and grasps around nothing

Fig. 8: Demonstration of the object prediction error issue. Small errors in object state prediction cause the model to veer off slightly from the desired trajectory and fail to correct itself, leading to task failure

pipeline requires the policy to generate additional observations to train the supervised predictor, this is not a significant issue as the generated observations are used for supervised learning rather than reinforcement learning, which empirically requires much less time. Furthermore, this approach could be made even more efficient by using the replay buffer as the source of these additional observations rather than generating them from scratch after training of the object state policy is finished.

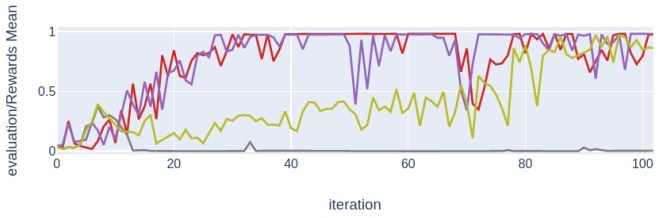


Fig. 9: Demonstration of sample efficiency gap between end to end approach and policy trained off of object state for the Reach task. The red and purple lines represent the policy trained off of object state with two different seeds, and the yellow and gray lines represent the end to end approach with two different seeds

VI. Conclusion and Future Work

While the approach explored in this paper does show some promise of closely imitating the paths of policies trained purely on object state as evidenced qualitatively from video rollouts, it does suffer from significant noise issues that make it difficult to train pixel policies that operate with the precision necessary to solve some robotics manipulation tasks. Common failure cases as object occlusion and the robot leaving the field of view make it difficult for any component to accurately predict object state using only a camera view because of incomplete information.

Because an issue with the precision of the model is dealing with particular failure cases given an image observation that does not adequately capture all information, it may be useful to augment the state passed to the object predictor to include extra information necessary to predict the object state. Rather than solely operating on pixel observations from the agentview camera, the object predictor could also be passed the robot sensor information, giving the robot's current position and orientation (which could be helpful in calculating parts of the object state that use the relative position of the robot). Furthermore, it may be useful to also pass in an alternative camera view to both help the robot better predict depth information using multiple views and allow for a larger field of view and avoid the case where the robot arm exits the field of view.

Furthermore, to decrease the error of the object predictor and allow for more precision in solving these robotic manipulation tasks, we could explore traditional techniques in supervised learning as using more advanced state of the art computer vision model architectures and data augmentations to the data passed to the model. These approaches could lead to a more accurate object predictor model overall and reduce the compound error issues seen during this study.

Finally, we could use an additional tuning process to avoid the compounding errors seen during training in the imitation learning approach by running a process like the DAgger algorithm, an imitation learning work that aims to correct covariate shift issues as the one seen in this paper [10]. This could include either generating many rollouts using the full testing pipeline, retraining the object predictor on these new rollouts, and repeating (as DAgger does) or running soft actor critic again but with the current trained object predictor and policy as an additional step to make the policy more robust to object predictor noise.

Once these issues are fixed, a future extension of this project to explore might be to use sim2real approaches as the ones explored by Rao et al. in [9] to determine how well these pixel policies actually translate in the real world and see if the technique is genuinely applicable for learning robust pixel policies for robots.

Acknowledgements

The author would like to thank Yuke Zhu and Zhenyu Jiang for running the course CS391R and providing the opportunity to do this project. The author would also like to thank the rest of the UT Austin Robot Perception and Learning Lab for providing a great learning environment for robotics reinforcement learning.

References

[1] Rail Berkeley. rlkit. https://github.com/rail-berkeley/rlkit, 2021.

[2] Linxi Fan, Guanzhi Wang, De-An Huang, Zhiding Yu, Li Fei-Fei, Yuke Zhu, and Anima Anandkumar. SECANT: self-expert cloning for zero-shot generalization of visual policies. *CoRR*, abs/2106.09678, 2021. URL https://arxiv.org/abs/2106.09678.

[3] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, and Tsuhan Chen. Recent advances in convolutional neural networks. *Pattern Recognition*, 77:354–377, 2018. ISSN 0031-3203. doi: https://doi.org/10.1016/j.patcog.2017.10.013. URL https://www.sciencedirect.com/science/article/pii/S0031320317304120.

[4] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. 2018.

[5] Arise Initiative. robosuite-benchmark. https://github.com/ARISE-Initiative/robosuite-benchmark, 2021.

[6] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z. Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *CoRR*, abs/1611.05397, 2016. URL http://arxiv.org/abs/1611.05397.

[7] Sascha Lange and Martin Riedmiller. Deep auto-encoder neural networks in reinforcement learning. pages 1–8, 2010. doi: 10.1109/IJCNN.2010.5596468.

[8] Michael Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. Reinforcement learning with augmented data. *CoRR*, abs/2004.14990, 2020. URL https://arxiv.org/abs/2004.14990.

[9] Kanishka Rao, Chris Harris, Alex Irpan, Sergey Levine, Julian Ibarz, and Mohi Khansari. Rl-cyclegan: Reinforcement learning aware simulation-to-real. *CoRR*, abs/2006.09001, 2020. URL https://arxiv.org/abs/2006.09001.

[10] Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. No-regret reductions for imitation learning and structured prediction. *CoRR*, abs/1011.0686, 2010. URL http://arxiv.org/abs/1011.0686.

[11] Aravind Srinivas, Michael Laskin, and Pieter Abbeel. CURL: contrastive unsupervised representations for reinforcement learning. *CoRR*, abs/2004.04136, 2020. URL https://arxiv.org/abs/2004.04136.

[12] Yuke Zhu, Josiah Wong, Ajay Mandlekar, and Roberto Martín-Martín. robosuite: A modular simulation framework and benchmark for robot learning. *CoRR*, abs/2009.12293, 2020. URL https://arxiv.org/abs/2009.12293.