Learning-free methods for Goal Conditioned Reinforcement Learning from Images

by

Alexander Van de Kleut

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2021

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

Supervisor(s):	Jeff Orchard Associate Professor, Dept. of Computer Science, University of Waterloo
Internal Member:	Kate Larson Professor, Dept. of Computer Science, University of Waterloo
Internal Member:	Pascal Poupart Professor, Dept. of Computer Science, University of Waterloo

Abstract

We are interested in training goal-conditioned reinforcement learning agents to reach arbitrary goals specified as images. In order to make our agent fully general, we provide the agent with only images of the environment and the goal image. Prior methods in goal-conditioned reinforcement learning from images use a learned lower-dimensional representation of images. These learned latent representations are not necessary to solve a variety of goal-conditioned tasks from images. We show that a goal-conditioned reinforcement learning policy can be successfully trained end-to-end from pixels by using simple reward functions. In contrast to prior work, we demonstrate that using negative raw pixel distance as a reward function is a strong baseline. We also show that using the negative Euclidian distance between feature vectors produced by a random convolutional neural network outperforms learned latent representations like convolutional variational autoencoders.

Acknowledgements

I would like to thank Dr. Jeff Orchard for his insight and guidance in completing this project. I would also like to thank Dr. Kate Larson and Dr. Pascal Poupart for agreeing to be members of the Examining Committee. I am thankful for the contributions and insights of Alex Werner and Brandon DeHart of the University of Waterloo's RoboHub for the opportunity to work with a physical robot.

Dedication

This is dedicated to my partner, who supported me endlessly throughout this project.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Reinforcement learning (RL) is a general learning framework that has gained significant popularity over the last decade. Recent successes include beating world champions at the video game Dota 2 [22], beating the world champion at the game of Go [25], and surpassing human baseline performance across a benchmark of 57 Atari games [2]. Reinforcement learning is attractive because of its generality; an agent in an environment learns to choose actions to maximize some cumulative reward. Any problem that can be framed this way can be approached with reinforcement learning.

In its original formulation, the agent gets some information about the current state of the environment, and it uses this to choose an action. The agent gets a reward indicating how good that action was given the state of the environment, and it has to learn how to choose actions that maximize the long-term cumulative reward. We can extend this framework to multi-goal reinforcement learning. In multi-goal RL, the input to the agent includes a specific goal to reach. The reward given to the agent by the environment reflects how close the agent is to reaching the goal. The agent then has to learn what actions are needed to reach the desired goal. Examples of problems that can be framed this way include robotic object manipulation, where a robot arm must manipulate an object to a desired goal configuration, and navigation problems, where the desired goal is some final position.

A fully general RL agent should be able to operate in a variety of environments, without learning from measurements specific to a particular task. This motivates learning directly from images. For multi-goal tasks, we present the agent with only an image of the goal and images of the environment as it acts. The difficulty of doing multi-goal reinforcement learning solely from images is defining the reward function that reflects how close an image is to the goal image.

Prior work has approached this problem using learned representations that map images to latent vectors. These approaches involve collecting data from the environment and solving some auxiliary task, like image reconstruction or predicting the number of timesteps needed to reach the goal. This requires learning additional parameters and tuning additional hyperparameters.

In this thesis, we show that learned latent representations are not necessary to solve some goal-conditioned tasks directly from images. We train an RL agent end-to-end directly from images, and define the reward to be the negative Euclidian distance between latent representations of the goal and current image. We consider various learning-free latent encodings, including raw pixel distance, random linear transformations, and random convolutional neural networks.

Chapter 2

Background

2.1 Reinforcement Learning

In reinforcement learning, we consider two entities: the agent and the environment. The environment is the world in which the agent lives and learns. At each time step t, the agent receives an observation of the state of the environment and uses this to choose an action. The agent uses this action to act on the environment, causing the environment to change. In response, the environment provides the agent with an updated observation and a reward. The reward is a scalar indicating how good or bad the current state of the environment is. The objective in reinforcement learning is to maximize the the cumulative reward received from the environment, which we call the return R.

As a simple example, consider the task of moving an object to a particular position using a robotic arm. The agent controls the robotic arm through actions, which change the position of the robot's end-effector. The observations include the current position of the robot's end-effector and the position of the object. The reward is the negative distance from the object's current position to the goal, so that being closer to the goal results in a higher reward.

When an observation contains a complete description of the state of the environment, we say the environment is fully observable and use the symbol s (for "state"). If the observation is instead an incomplete description of the state of the environment, we say the environment is partially observable and use the symbol o (for "observation"). In this thesis, we focus on partially observable environments.

An agent chooses actions by sampling from a policy $a \sim \pi(a \mid o)$, which is a probability distribution over actions conditional on the current observation. We often represent policies using neural networks with parameters θ and thus write π_θ for the policy.

The agent repeatedly experiences observations o, chooses actions a, and receives the next observation o' and reward $r = r(o, a, o')$. We call this a state transition and represent it as a tuple (o, a, r, o'). A collection of state transitions experienced by the agent is called a trajectory τ. The goal of the agent is to learn parameters θ so that the agent maximizes the expected return over all trajectories collected under the policy π_θ

$$\mathbb{E}_{\tau\sim\pi}\left[R(\tau)\right].$$

In general we assume that a policy may act forever in its environment, so the return $R(\tau)$ may be infinite. To account for this, we maximize the infinite-horizon discounted return

$$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t, \tag{2.1}$$

where $\gamma \in (0, 1)$ is a discount factor that ensures the return is finite. Larger values of γ weigh future rewards more heavily than smaller values of γ.

We can measure how good an action a is given an observation o using the action-value function. Given some policy π, the action-value function $Q^\pi(o, a)$ is the expected return assuming we start the trajectory with an observation o and initially choose an action a, following the policy π every step thereafter. We define $Q^\pi(o, a)$ as

$$\begin{aligned} Q^\pi(o, a) &= \mathbb{E}_{\tau\sim\pi}\left[R(\tau) \mid s_0 = s, a_0 = a\right] \\ &= \mathbb{E}_{\tau\sim\pi}\left[r_t + \gamma \mathbb{E}_{a'\sim\pi}\left[Q^\pi(o', a')\right] \mid s_0 = s, a_0 = a\right], \end{aligned}$$

where the expansion inside the expectation can be derived from the definition of the infinite-horizon discounted return (2.1). When π is the optimal policy π^* that maximizes this return over all trajectories, we call this the optimal action-value function $Q^*(o, a)$, which we can write as

$$Q^*(o, a) = \max_\pi \mathbb{E}_{\tau\sim\pi}\left[R(\tau) \mid s_0 = a, a_0 = a\right].$$

2.1.1 Soft Actor-Critic (SAC)

In this thesis, we use soft actor-critic (SAC), which interleaves learning an approximator to Q^* and an approximator to π^*.

The critic Q_ϕ is a neural network with parameters ϕ that approximates Q^*. It uses its own predictions to improve itself and better approximate Q^*.

We can recursively expand the return $R(\tau)$ in the optimal action-value function to obtain

$$\begin{aligned} Q^*(o,a) &= \max_\pi \mathbb{E}_{\tau\sim\pi}\left[R(\tau) \mid s_0 = a, a_0 = a\right] \\ &= \max_\pi \mathbb{E}_{\tau\sim\pi}\left[r_t + \gamma \max_{a'} Q^*(s', a') \mid s_0 = a, a_0 = a\right]. \end{aligned}$$

This general form is known as the Bellman optimality equation. Given our approximator Q_ϕ, we can measure how closely it satisfies Bellman optimality by measuring the squared-error between the two definitions. Given some state transition (o, a, r, o') we can define the loss

$$\begin{aligned} \mathcal{L}(\phi) &= (Q_\phi(o,a) - y)^2 \\ y &= \left(r + \gamma \max_{a'} Q_\phi(o', a')\right) \end{aligned}$$

where y is treated as a constant regression target. The max over a' is approximated by sampling directly from the current policy $a' \sim \pi_\theta(a' \mid o')$. By minimizing $\mathcal{L}(\phi)$, Q_ϕ approaches Bellman optimality. We maintain a collection of state transitions in a replay buffer $\mathcal{D}$ and compute $\mathcal{L}(\phi)$ over a minibatch of state transitions sampled randomly from the replay buffer.

We optimize the policy by sampling observations o from the replay buffer, computing actions $a \sim \pi_\theta(a \mid o)$, and plugging them into Q_ϕ. Since Q_ϕ approximates the expected return that we want to maximize, we can treat $Q_\phi(o,a)$ as the quantity to maximize, while keeping the parameters ϕ constant:

$$J(\theta) = \mathbb{E}_{\substack{a\sim\pi_\theta \\ o\sim\mathcal{D}}}\left[Q_\phi(o,a)\right].$$

For this to work, we assume that sampling an action a is differentiable with respect to the policy parameters θ. When the environment expects actions that are real-valued vectors, we typically represent a stochastic policy by a diagonal Gaussian distribution. Take $\mathcal{N}(\mu_\theta(o), \sigma_\theta(o))$ to be the diagonal Gaussian distribution produced by the policy π_θ. Directly sampling from this distribution is a nondifferentiable operation with respect to the policy parameters θ. However, we can reparametrize the sampling operation to make it differentiable by computing the action as

$$\begin{aligned} &a = \mu_\theta(o) + \sigma_\theta(o) \odot \epsilon \\ &\epsilon \sim \mathcal{N}(0, I). \end{aligned}$$

We make the nondifferentiable sampling operation disconnected from the policy parameters by sampling ϵ from a standard diagonal Gaussian distribution, and thus make $Q_\phi(o, a)$ differentiable with respect to θ. See figure 2.1 for a simplified diagram of the architecture. SAC encourages long-term exploration by adding an entropy bonus to the reward. The

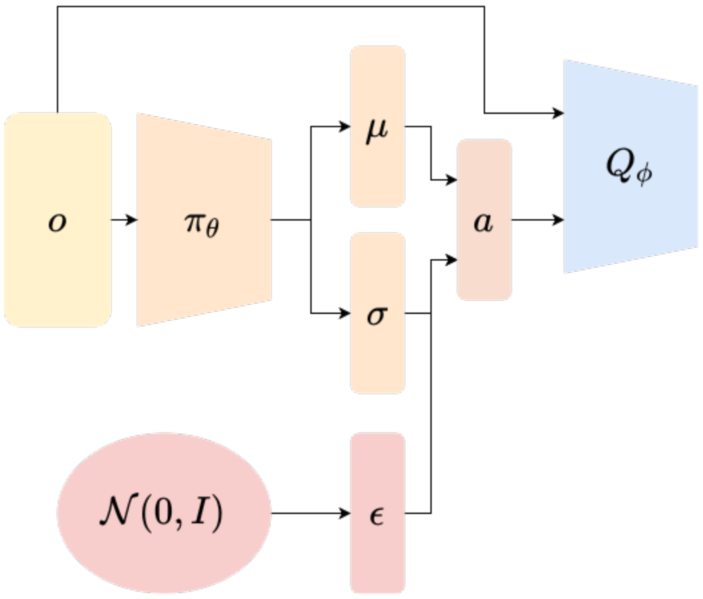

Figure 2.1: Soft Actor-Critic computation graph.

entropy bonus $H(\pi(a \mid o))$ discourages the policy from converging to a suboptimal policy too quickly and encourages a more diverse choice of actions. SAC uses two critics and takes the minimum of their predictions to generate targets y. It also uses copies of the critics to generate targets, whose weights are updated via exponential averaging to smooth out targets.

We use SAC in this thesis because it is sample efficient. SAC is an off-policy RL algorithm, which means that it can learn from any valid state transition (o, a, r, o'). This means that it can reuse data collected from an earlier version of the policy to optimize the current policy. It also differs from many other RL algorithms by performing a parameter optimization step at every environment step. These two aspects combined mean that SAC can learn a good policy from very little data. This is especially important in the robotics regime, where environment interaction and data collection becomes the bottleneck.

2.2 Goal-Conditioned Reinforcement Learning from Images

2.2.1 Image Observations

Learning from images has been an important part of deep reinforcement learning since deep Q-learning (DQN) was first described by Mnih *et al.* in 2015 [18]. Because learning policies directly from pixels and rewards is a very general approach, DQN quickly popularized deep reinforcement learning. However, learning directly from images can sometimes require hundreds of millions of environment steps to learn a successful policy.

In simulated RL environments, it is possible to know the exact state of different entities in the environment. However, in the real world, knowing the exact state of the environment is impractical without extensive instrumentation. By learning from images, we provide the agent with a single source of information which does not require additional instrumentation, and which can be easily implemented in the real world using a camera.

Learning from images also avoids need for a combinatorial representation of the environment. For example, in an environment with one, two, or three objects, how would we represent the state when some of the objects are missing? By using images, we have fixed-size observations that can encode a variable number of objects [20, 29].

Learning from images can be difficult because of partial observability. If an agent is only getting images of the environment as inputs, these images may not contain the information necessary to completely describe the environment. For example, an image does not contain information about velocities or accelerations, which may be required to solve the task at hand. Other parts of the environment may be occluded or concealed. Thus, the agent needs to learn how to act with limited information.

2.2.2 Observations and Partially Observable Environments

Here we formalize the problem of learning from images. In fully observable environments, the agent gets access to a complete description of the environment. We say that the agent receives states $s \in \mathcal{S}$ where $\mathcal{S}$ is the state space. On the other hand, in partially observable environments, the agent instead gets some alternative impoverished representation of the state, such as images. We say the agent receives observations $o \in \mathcal{O}$ where $\mathcal{O}$ is the observation space.

In the case of images, $\mathcal{O}$ is the space of valid images of the environment. We imagine there is some function $f_o : \mathcal{S} \to \mathcal{O}$ that generates an observation given a state $o = f_o(s)$. See figure 2.2 for a visual representation of the difference between an observation space and a state space. Partial observability makes learning challenging because the agent cannot necessarily know what the outcome o' of choosing an action $a \sim \pi(a \mid o)$ will be, since environment dynamics are described in state space and not in observation space.

If we want to define a problem using reinforcement learning, we need to define both the agent and the environment. To define a partially observable environment for reinforcement learning, we need to define a reward function. Ideally we would just compute the reward directly from observations as $r(o, a, o')$ (rather than $r(s, a, s')$ when we have access to the underlying states). RL researchers typically "cheat" in simulation by computing the reward based on the underlying state while only showing the observations to the agent. This still requires knowing the state, even if the agent doesn't get to know it. In the real world, the state space $\mathcal{S}$ includes all of the physical parameters needed to model the environment dynamics, which we may not know without extensive instrumentation. For example, an agent learning to solve a maze may not know its true distance to the goal without some way of measuring it. Truly learning from images requires computing a reward function directly from images.

2.2.3 Goal-Conditioned RL

In the normal description of reinforcement learning, we specify a single reward function $r(s, a, s')$ that represents how good a state transition (s, a, s') is. Maximizing this reward should correspond to solving the task at hand. One drawback to this is that the agent is only capable of solving a single task specified via the reward function. What if we want the agent to be able to solve multiple tasks? This motivates the definition of goal-conditioned RL. In goal-conditioned RL, the reward function is additionally conditioned on a goal g, which is sampled from some goal space $\mathcal{G}$. The reward function now depends on the goal $r(s, a, s', g)$ and should specify, in some sense, how close the agent is to reaching that goal. This allows RL to scale to settings with multiple related tasks, such as navigating in a room or manipulating objects [8].

We can think of the goal space $\mathcal{G}$ as being a representation of the state space $\mathcal{S}$ where it is easy to specify how close we are to the goal. For example, consider the task of moving a robotic arm so that the end-effector reaches a certain (x, y, z) coordinate. It makes sense to describe the goals in terms of those (x, y, z) coordinates. However, the actual underlying state space $\mathcal{S}$ may instead only contain the angles and velocities of each of

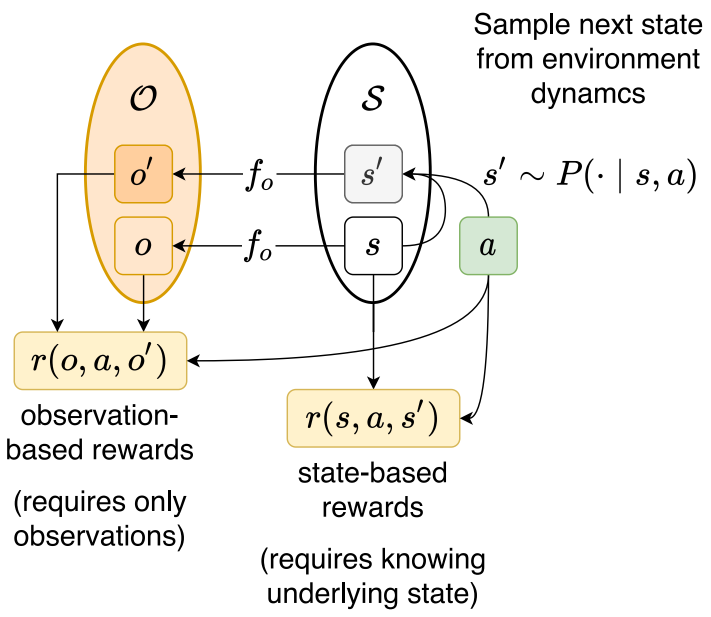
Sample next state from environment dynamcs
$s' \sim P(\cdot \mid s, a)$
$\mathcal{O}$
$\mathcal{S}$
o'
o
f_o
f_o
s'
s
a
$r(o, a, o')$
observation-based rewards
(requires only observations)
$r(s, a, s')$
state-based rewards
(requires knowing underlying state)

Figure 2.2: Generating a transition in a partially observable environment. In simulation, we can compute the reward based on the underlying states s and s'. In the real world, we don't have access to the underlying state of the environment. To do RL in the real world, we would need a reward function defined instead in terms of the corresponding observations o and o', which may represent pixels of an image.

the robot's joints. The state space does not immediately contain information about the (x, y, z) coordinates of the end-effector. We imagine some function $f_g : \mathcal{S} \to \mathcal{G}$ that maps a state s to its corresponding representation in goal space. This is exactly analogous to the function f_o that maps states to observations. Note that, like observations, there can be many states s that map to the same goal g. See figure 2.3 for a schematic. Continuing the example given above, there may be many robot configurations with a given end-effector position.

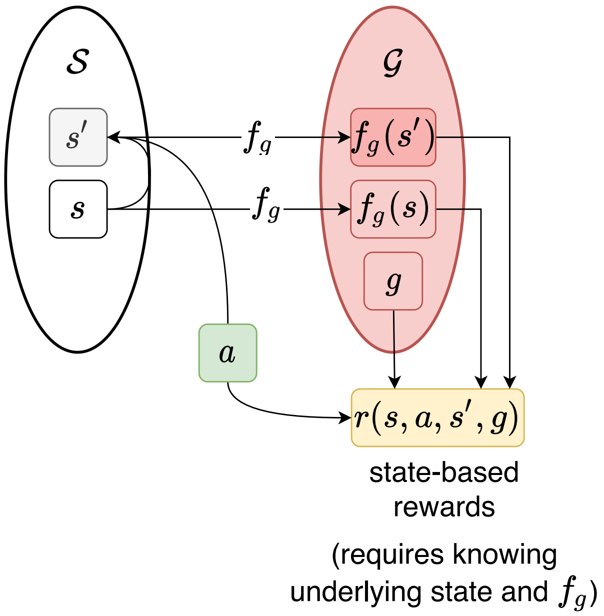

Figure 2.3: Generating a transition in the goal-conditioned context. The agent sees the current state s and a goal g and chooses an action a, producing a state s'. We project s and s' into goal space to compute the reward with respect to the goal. It is common to take $r(s, a, s', g) = -\|f_g(s') - g\|_2$.

When states and goals are continuous, reaching a state s such that $f_g(s) = g$ exactly can be very hard. Instead, it is common practice to say that a goal has been "reached" if a state s belongs to the preimage S^g of some ϵ-ball around the goal g

$$S^g = \{s \in \mathcal{S} : \|f_g(s) - g\|_2 \leq \epsilon\}.$$

For example, if a robot needs to navigate to a particular location, we may say the goal is reached when it is within 0.5m of the goal.

Given a transition (s, a, s'), one way to specify the reward $r(s, a, s', g)$ is to simply map the next state s' to its corresponding goal and take the negative Euclidian distance to the goal to be the reward

$$r(s, a, s', g) = -\|f_g(s') - g\|_2.$$

This way, the reward is at a maximum of 0 when the goal is reached at s'.

2.2.4 Solving Goal-Conditioned Tasks from Images

This thesis focuses on environments that are both goal-conditioned and use image observations. The goal-conditioned aspect allows us to arbitrarily specify tasks to the agent at evaluation time, and the image aspect allows us to specify these tasks by just showing the agent an image of the desired environment configuration. For example, a navigation task can be specified to the agent by just showing it where you want it to end up.

Normally the reward is computed using the underlying state of the environment. For example, if the task involves manipulating objects with a robot, the reward may be computed based on the position of the object and its desired goal. However, when using images as observations, this information is not directly accessible, so we need to ensure our reward function is computable directly from images. See figure 2.4, which builds upon figures 2.2 and 2.3 for a diagrammatic representation of goal-conditioned RL from images.

In the goal-conditioned context, computing rewards amounts to determining how close o' is to o_g directly from raw pixels. It is not easy, in general, to determine how "close" an image observation is to a goal image. Furthermore, it is possible that only some aspects of the image correspond to the task at hand. Consider the task of moving an object to a specified location using a robotic arm. Only the pixels corresponding to the object actually matter for solving the task, but those pixels may only make up a small portion of the image.

One of the main focuses of this thesis is evaluating reward functions $r(o, a, o', o_g)$ that work directly from image observations and image goals.

2.3 Related Work

2.3.1 Vision-based RL

In recent years, we have seen steady progress in state-of-the-art (SOTA) performance for RL tasks from images. In particular, we have seen significant progress in sample efficiency.

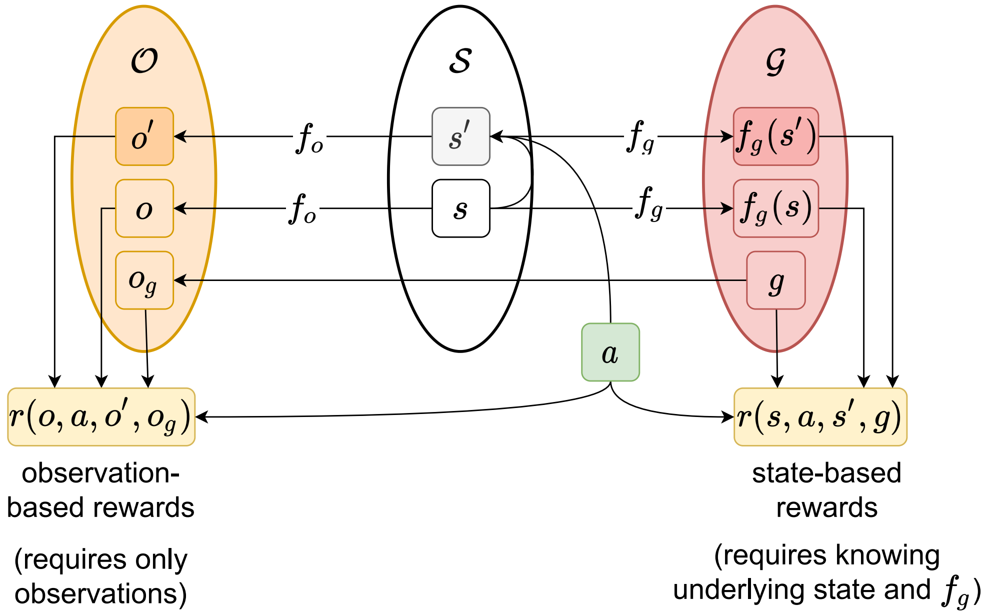

Figure 2.4: Generating a transition in a partially observable goal-conditioned environment. The environment is in a state s with a goal g. The agent sees the corresponding observation and goal observation o and o_g and chooses an action a. The environment transitions to the next state s' and a corresponding observation o' is generated. In simulation, we can compute a reward $r(s, a, s', g)$ directly from the underlying states. In the real world we only have access to (o, a, o', o_g) and therefore need to be able to compute the reward directly from the observations and goal observation.

Sample efficiency can be thought of as agent performance per number of environment interaction steps.

PlaNet [12], and its successor Dreamer [11], held SOTA on the DeepMind Control Suite, a series of simulated continuous control problems from images [26]. PlaNet and Dreamer are model-based methods. They convert an image observation o_t at time step t into a latent representation z_t, and learn a forward dynamics model to predict the next latent representation z_{t+1} given z_t and the action a_t. These methods achieve good sample efficiency by generating artificial training data given a limited set of data collected from the environment. While the agent may only collect 10^5 transitions from the environment, it may simulate additional trajectories using the forward dynamics model, generating more training data for the policy. PlaNet and Dreamer are very similar, differing mostly in the models used for encoding images and doing forward dynamics modelling. These methods are complex and require expertise to understand and implement. Their sample efficiency comes from the learned dynamics model, which is used to generate additional training data, and does not count towards the actual number of environment steps used to measure sample efficiency.

Srinivas *et al.* [15] also learn sample-efficienct policies from images. However, instead of using a model-based approach to gain sample efficiency, they instead focus on learning a suitable image representation. The authors use contrastive unsupervised representation learning (CURL) to learn a representation of input images, and then use that learned representation to train an RL algorithm. In their method, they randomly crop two portions of the input image and generate two corresponding features vectors using a convolutional neural network. Then, the network is trained to produce the same features for both cropped inputs so that the output is invariant to random cropping. Once trained, this convolutional neural network is used to convert all image observations o to latent vectors z, and an RL agent is trained using SAC on top of these latent representations. At the time of its publication, CURL outperformed Dreamer in terms of sample efficiency and final performance on the DeepMind Control Suite.

A key idea from CURL is to take an image, perform image transformations (random crops), and train the network to produce the same output for both versions of the image. This idea was distilled and then expanded upon by Kostrikov *et al.* [14]. Rather than having the convolutional encoder trained using some auxiliary task (like in CURL), the encoder is trained end-to-end with the critic. However, both targets y and predictions $Q_\phi(o, a)$ are generated using transformed versions of o. They transform images using random horizontal and vertical shifts by up to 4 pixels. This way, the critic is trained to make the same predictions despite slight variations in the input image. The data transformation technique both provides additional training data to the critic and makes predictions more robust, quickly allowing the critic's convolutional encoder to learn a good representation of image observations. The policy shares the convolutional encoder with the critic, but does not update its parameters. This method is called data-regularized Q-learning (DrQ) and supersedes CURL in sample efficiency and final performance.

Concurrently, Laskin *et al.* [16] used image transformation to achieve SOTA performance on RL tasks from images. They examined many different kinds of image transformation techniques, but came to the same conclusion as Kostrikov *et al.* that random crop and random translation provide the best improvement in performance. They call this method RL with transformed data (RAD). Like DrQ, RAD transforms image observations before passing them to the critic during training. However, unlike DrQ, only a single transformation is applied to generate targets and predictions. Both DrQ and RAD achieve similar performance on the DeepMind Control Suite.

We can see that parsimonious ideas like DrQ and RAD can outperform complex methods like Dreamer and PlaNet. These methods focus on perception by learning a good latent representation of images such that doing RL on top of these latent representations is easy. In this thesis, we adopt two ideas from DrQ and RAD. First, we use the same training regime and architecture as DrQ, where the critic and encoder share a convolutional encoder and only the critic's loss gradients pass through the convolutional encoder. Second, we adopt the idea of using convolutional neural networks to generate latent vectors from image observations. See figure 3.1 for a simplified diagram of the architecture.

2.3.2 Goal-Conditioned RL from Images

Visual Reinforcement Learning with Imagined Goals (RIG) was one of the first papers that dealt directly with goal-conditioned RL from images [20]. RIG proposes learning a low-dimensional latent representation from images using a variational autoencoder. The RL agent is trained on top of this latent representation, and rewards are computed as negative latent distance to the goal.

Briefly, a variational autoencoder is a neural network composed of an encoder e and a decoder d. The encoder e maps an observation o to a latent representation z, and the decoder maps the latent representation z back to an approximation of the input $\hat{o}$.

The encoder outputs the mean $\mu(o)$ and log-standard deviation $\log \sigma(o)$ of a diagonal Gaussian distribution. Then, a latent vector is generated by sampling from that distribution $z \sim \mathcal{N}(\mu(o), \sigma(o))$. The decoder is trained to minimize the reconstruction loss $\mathbb{E}\left[\|o - d(z)\|\right]$ given a latent vector z.

We encourage the encoder to output $\mu(o)$ and $\log \sigma(o)$ corresponding to a standard diagonal Gaussian distribution to ensure a continuous latent space. This is accomplished this by minimizing the KL-divergence between the output distribution $\mathcal{N}(\mu(o), \sigma(o))$ and $\mathcal{N}(0, I)$.

Given a collection $\mathcal{D}$ of image observations, we define the VAE loss as

$$\mathcal{L}(\mathcal{D}) = \mathbb{E}_{o \sim \mathcal{D}} \left[\underbrace{\|o - d(z)\|^2}_{\text{Reconstruction loss}} + \overbrace{\beta \mathbb{KL}\left[\mathcal{N}(\mu(o), \sigma(o)) \parallel \mathcal{N}(0, I)\right]}^{\text{KL Divergence loss}} \right]$$

Nair *et al.* [20] collect image observations to use for training the VAE by rolling out a randomly acting policy for some number of time steps in the environment. They then train a policy π_θ and critic Q_ϕ entirely in the latent space $\mathcal{Z}$ using the encoder of the VAE. In their experiments, they show that RIG achieves good performance, sometimes on par with an "oracle" method which uses the ground truth states as input instead of latent vectors. In this thesis, we use the encoder from a VAE trained on images of the environment to generate a reward function. We take the negative Euclidian distance between the latent encoding of the observation and the latent encoding of the goal

$$r(o, a, o', o_g) = -\|e(o') - e(o_g)\|_2,$$

so that the reward reaches a maximum of 0 when $o' = o_g$.

Using VAEs as a method of encoding images into a low-dimensional latent space is a very common technique [10, 20, 19, 21, 4]. Successors to RIG have modified the VAE architecture to account for different types and numbers of objects to manipulate [19] and to do model-based RL [21].

VAEs are generally trained on images obtained by a randomly-acting policy, which are collected before training starts. The problem with this approach is that random actions may be unable to sufficiently explore the environment, limiting the kinds of observations the VAE is trained on. When the agent observes an image the VAE was not trained on, the corresponding latent vector z may not be meaningful. This is especially detrimental to methods like RIG that use these latent vectors as input to the policy and the critic. The only attempt to limit this problem that is mentioned in the literature is to fine-tune the VAE on additional training data from the replay buffer during training. However, this will change the distribution of inputs to the policy and the critic, which might be challenging for the agent to overcome.

The VAE latent space represents one family of approaches to goal-conditioned RL from images in the literature. Another approach is to learn an encoding ϕ such that the distance between latent vectors corresponds to some measure. For example, Florensa *et al.* [7] learn an encoding such that distance between latent vectors corresponds to the minimum number of timesteps to go from one image observation to another in the environment.

Ghosh *et al.* [9] define the idea of actionable distance between two observations. From any current observation o, they treat two observations o_1 and o_2 as goal observations for some trained goal-conditioned policy. They define a symmetric distance function that describes how different the policy outputs are with o_1 as a goal and with o_2 as a goal by defining the actionable distance

$$D_{\text{Act}}(o_1, o_2) = \mathbb{E}_o\Big[\mathbb{KL}\big[\pi_\theta(a \mid o, o_1)\|\pi_\theta(a \mid o, o_2)\big] + \mathbb{KL}\big[\pi_\theta(a \mid o, o_2)\|\pi_\theta(a \mid o, o_1)\big]\Big].$$

By this distance, two observations o_1 and o_2 are close if a goal-conditioned policy chooses similar actions on average with o_1 as the goal versus with o_2 as the goal. Likewise, o_1 and

o_2 are distant if a goal-conditioned policy chooses different actions on average with o_1 as the goal versus with o_2 as the goal.

Given this distance function and a trained goal-conditioned policy, they train a neural network ϕ to produce latent vectors so that Euclidian distance corresponds to actional distance

$$\|\phi(o_1) - \phi(o_2)\|_2 = D_{\text{Act}}(o_1, o_2)$$

This encoding ϕ produces a latent space $\mathcal{Z}$ where Euclidian distance corresponds to functional difference between states. Euclidian distance in latent space here is optimized to have a particular meaning, whereas the Euclidian distance in the latent space generated by a VAE is not inherently meaningful.

Wulfmeier *et al.* [28] recently examined three types of learned representations used in robotic manipulation tasks. They compared VAEs with two other models: Transporter and MONet. These two models learn a representation from a series of images that focuses on objects that can be manipulated. Transporter tries to decompose an image into a series of coordinates along the image axes, with each set of coordinates representing an object in the image. MONet is a modified VAE that tries to decompose an image into a predetermined number of regions using a recurrent attention mechanism. They also use an automatic colour thresholding technique to segment input images by colour. The authors find that MONet and Transporter are capable of learning "disentangled" representations of image observations such that each dimension in the latent representation corresponds to some semantically meaningful element of the image observation. Such semantically meaningful elements might include object colour, shape, size, or position.

These methods perform well on the given tasks, but require a significant amount of manual hand-engineering to get good performance. Many hyperparameters for these representation learning methods need to be chosen in advance, such as how many objects to track for Transporter and how many regions to attend to when using MONet. Furthermore, these representation learning methods are trained using offline data collected by a trained goal-conditioned policy. This is necessary to give these representation learning methods sufficiently variable data to learn good representations. However, this is limiting, because it requires that we already have an agent capable of solving the task in order to train another agent capable of solving the task. Also, the authors decompose complex tasks like lifting, stacking, and pushing into sequentially-presented sub-tasks. For example, to get an agent to lift a block, they first present the "reaching" goal, then the "grasping" goal, and finally, the "lift" goal, rather than just presenting the final goal position. As such, their results are highly specific to their simulation environment and do not generalize well.

2.3.3 Random Convolutional Neural Networks

In this thesis, we present randomly initialized convolutional neural networks as a viable method of generating image encodings. This avenue of research appears to be relatively unexplored, especially in the reinforcement learning setting.

Random Network Distillation (RND) [5] was a novel approach to exploration in reinforcement learning using image observations. It demonstrated the power of randomly initialized convolutional neural networks for feature generation. The authors initialize two convolutional neural networks with the same architecture and different random initializations. The first one has frozen parameters and is called the "target network" and the second one has learnable parameters and is called the "predictor network".

When the agent sees an image observation, the target network generates a latent encoding of that image by doing a forward pass. The predictor network does the same. The predictor network is then trained to produce the same output as the target network by minimizing the mean-squared-error (MSE) between the target and the prediction via gradient descent.

For images that the agent has seen many times (and thus trained on many times), we can expect the MSE to be low. However, for images that are new or rare for the agent to see, we expect these images to be outside the training distribution for the predictor network, and thus expect the MSE to be higher. Thus, the MSE for predicting random features of a convolutional neural network can be used as a proxy for observation novelty. This novelty can be used as a reward bonus to encourage the agent to explore. The importance of this paper for our purposes is that it shows that random features from a convolutional neural network contain sufficient features from the original image to solve a difficult RL task.

Ulyanov *et al.* [17] experimented with using randomly initialized convolutional neural networks as image priors for inverse problems like image denoising, super-resolution, and image inpainting. The authors use a convolutional neural network $f_\theta(z)$ to transform some randomly initialized fixed vector z from an arbitrary latent space to image space using a series of convolution operations, upsampling, and nonlinear activation functions. The authors find that the sequential application of convolution, upsampling, and nonlinear activation is sufficient to capture the kind of relationships between pixel neighbourhoods that we also see in natural images. Randomly initialized convolutional neural networks can also capture some of the low-level statistics of natural images, in particular local and translation-invariant aspects of the image. Furthermore, the fact that the convolution filter is applied across the entire visual field imposes a certain stationarity on the output of the convolution layers.

Traditional approaches to denoising, super-resolution and inpainting use regularization on the generated image to encourage natural-looking images, such as the total-variation norm. The authors find that convolutional architectures provide enough structure to the output to encourage this kind of natural appearance. This work demonstrates that the structure of a randomly initialized convolutional neural network is enough to extract meaningful information about neighbourhoods of pixels in an image.

Chapter 3

Methods

3.1 Experimental Approach and Baselines

We consider the problem of training a goal-conditioned policy using only image observations. Generally, previous work used a pre-trained encoding $\phi : \mathcal{O} \to \mathcal{Z}$ that mapped all images and goals to latent vectors, and then trained an RL agent entirely in the corresponding latent space. Instead, we choose to train our agent end-to-end from raw pixels. We only use the latent space $\mathcal{Z}$ for computing rewards as the latent distance between the next observation and the goal

$$r(o, a, o', o_g) = -\|\phi(o') - \phi(o_g)\|_2.$$

We compare several methods for the form of ϕ:

- **Random Linear Features (LINEAR)**: We can use a random linear projection to map from observations to latent vectors. This allows us to do random dimensionality reduction without learning, and allows us to determine the overall importance of simply reducing the dimensionality of observations.

- **Random Convolutional Features (CONV)**: This method is the main approach of this thesis. We use a small, randomly initialized convolutional neural network followed by a single fully-connected layer to produce latent vectors. Like LINEAR, CONV uses an untrained randomly initialized neural network to do dimensionality reduction. However, unlike LINEAR, CONV uses a sequence of convolution operations which compute spatial information. In contrast, LINEAR does not use spatial information. We use the convolutional architecture from Kostrikov *et al.* [14]. This is also the same architecture used for the convolutional encoder by all RL agents.

- **β-Variational Autoencoder (VAE)**: The VAE is a popular model for embedding images into a latent space. As discussed above, it is an unsupervised learning technique to learn a low-dimensional latent representation of image observations. The VAE must be trained on data collected from the environment. Like other previous approaches, we collect data using an agent that selects actions uniformly randomly for a fixed number of timesteps. Then the VAE is trained on that data for a given number of epochs, using some value for β and some learning rate. We use the architecture from Nair *et al.* [20]. The encoding is given by $\phi(o) = e(o)$ where e is the encoder of the VAE. Like in RIG, we first flatten image observations before using them as inputs to the VAE, so the VAE uses only fully-connected layers [20].

- **Convolutional Variational Autoencoder (CVAE)**: This is a variational autoencoder that makes use of convolutional layers for encoding, and transposed convolutional layers for decoding. It is trained in the same way as a VAE, but uses a different architecture.

- **VGG**: We use the convolutional layers of a VGG convolutional neural network trained to classify images from the ImageNet dataset. The idea is that the convolution kernels learned by this network should be useful for image processing in general.

- **Raw Pixels (PIXEL)**: We compare against the baseline of performing no encoding of images at all, except to flatten images into vectors and normalize pixel values to be between 0 and 1. By using pixel distance as a reward, we evaluate the actual difficulty of solving the task from pixels, and determine the necessity of having an encoding scheme at all.

3.2 Architecture

Here we describe our model architecture, adapted from Kostrikov *et al.* [14] to work with goals. Our model architecture is shown in figure 3.1.

The image observation o and the goal observation o_g are used to compute the latent reward using some latent encoding ϕ. The observation and goal are also passed to the shared convolutional layers of the policy and the critic. We reuse the convolutional encoder on both the observation and the goal, and concatenate the resulting output. This concatenated output is used as a combined representation for both the policy and the critic.

To update the policy, we sample an observation and goal from the replay buffer $\mathcal{D}$, compute the concatenated output from the shared convolutional layers, and treat it as a constant input to the policy so that gradients cannot flow back into the shared convolutional layers. We then choose an action based on this input, and pass the concatenated output and action to the critic, whose output is used to sample the objective $J(\theta) = Q_\phi(o, a, o')$. We then maximize $J(\theta)$ via gradient ascent on only the policy parameters θ, treating the parameters ϕ of Q_ϕ as constant. We use orange lines to represent the computation path along which the policy objective gradients flow.

To update the critic, we sample an observation, goal, and action from the replay buffer. We compute the concatenated output from the shared convolutional layers, and pass this along with the sampled action to the critic. We then compute the target y for the critic to regress onto using the latent reward. We then minimize $\mathcal{L}(\phi)$ via gradient descent on the parameters ϕ of Q_ϕ as well as on the parameters of the shared convolutional layers. We use blue lines to represent the computation path along which critic loss gradients flow.

To simplify the diagram, we do not show the additional action and next state sampled from the replay buffer, nor do we show the additional critic and target critics used by SAC to improve performance.

3.3 Environments

We consider three robotics manipulation tasks that use the MuJoCo physics simulator [27]. These environments allow us to train and compare many agents in parallel while running faster than real-time physics. In each environment, the robot is controlled via positional control so that actions correspond to changes in (x, y, z) coordinates of the end-effector. This is easier than learning to control motor torques directly and avoids the need for recurrent network architectures or using previous experience frames as additional inputs to the agent to account for velocity and acceleration.

The agent only gets access to the current observation image and the goal image. The agent gets up to 50 timesteps to solve the task before the environment is reset. We use simulator states for evaluation. We generate goal images by setting the environment to the goal state, rendering an image, and resetting the environment to its initial state. See figure 3.2 for images of the environments.

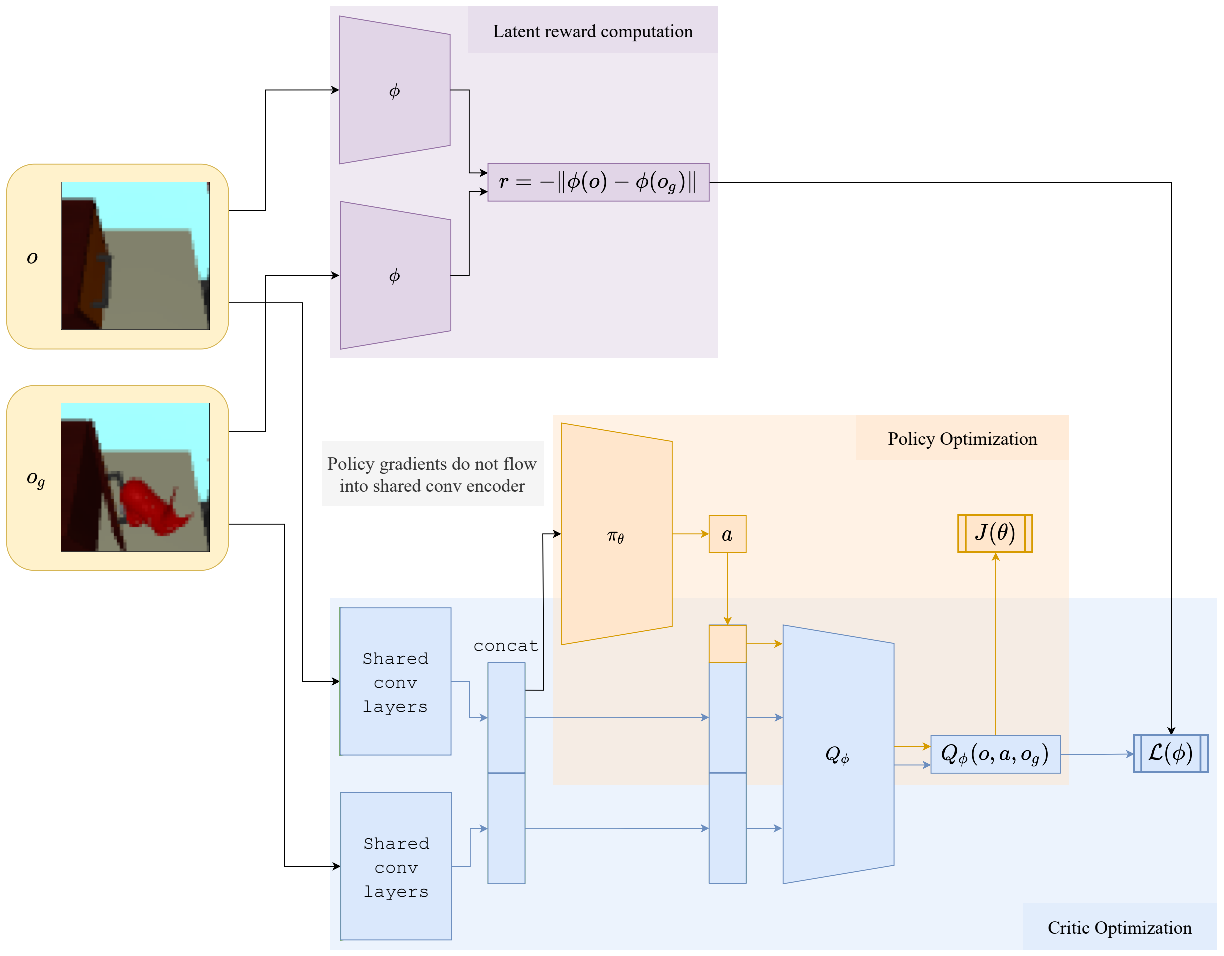

Figure 3.1: A simplified diagram of the model architecture, showing how observations and goals are used to compute rewards, and showing the computation graph for the policy and critic optimization. Policy optimization gradients flow through orange edges of the computation graph and critic optimization gradients flow through blue edges. This diagram is not an exact reflection of our architecture due to the complexity of the SAC learning algorithm.

Two environments are released as part of the standard OpenAI gym package [23] and one is part of the MultiWorld suite used by RIG and successors [20].

1. `FetchReach-v1`: The goal is to move the robot's end-effector to a specified (x, y, z) coordinate. The goal position changes with each episode. When operating from states, this task is easy, because the correct action is just the difference between the goal position and the current position. Images are 84×84 pixels, following standards used by RAD and DrQ [16, 14]. Success in the task is defined as moving the end-effector to within 0.05 metres of the goal.

2. `SawyerDoor-v1`: The robot must learn to grasp a door handle and then move the door to a desired angle. The agent is evaluated both in how close the door angle is to the desired position, as well as how close the end-effector of the robot is to the desired position. Images are 48×48 pixels in accordance with prior work using these environments, like RIG [20]. Success in the task has two components. Success for the door position is defined as being within 0.02 radians (1.14 degrees) of the goal, and success for the robot arm positions is defined as being within 0.03 metres of the goal.

3. `FetchPush-v1`: The goal is to push an object to a desired goal position. The goal position changes with each episode. This task is much harder than `FetchReach-v1` because the agent needs to learn to move the arm to an appropriate position to be able to push the object. This task is difficult to solve directly from pixels, because reaching the goal involves controlling only a small subset of pixels. Success in the task is defined as moving the object to within 0.05 metres of the goal.

Since each of our environments are simulated, we can evaluate an agent based on the ground truth states and goals. While the agent and reward function only ever get access to images, we can compute, for example, the true distance of a robot's end-effector to the goal position by querying the simulator. Such evaluation is difficult or impossible in real life without extensive instrumentation.

3.4 Hyperparameter Optimization

Every reinforcement learning algorithm makes use of several hyperparameters. There are several ways that hyperparameters are chosen in reinforcement learning research:

- Hyperparameters can be copied over from prior work. This is especially common for neural network architectures, which are often chosen heuristically or tuned for one paper, and then duplicated in subsequent papers. In this thesis, we use the convolutional architecture defined by Kostrikov *et al.* [14].
- Hyperparameters can be chosen heuristically or left as default values. Certain algorithms, like SAC, have tunable hyperparameters with default values. Often a good algorithm will require little or no tuning of these hyperparameters to perform well.
- Hyperparameters can be tuned using some search algorithm. The majority of the hyperparameters used in this thesis are tunable and interact to impact the performance of the RL agent. A full list of hyperparameters and their ranges for searching is available in appendix A.1.

Unfortunately, it is a common practice in the literature to not disclose how hyperparameters were chosen for a given method. It is possible that the authors tried hundreds of hyperparameter configurations before finding a configuration that worked well on the problem. Furthermore, it is a common practice to then assume that the hyperparameters that were tuned for the proposed method will work equally well for the baseline comparison methods. However, this is an unfair procedure, because it allocates the entire hyperparameter optimization budget onto one method, while not affording the same hyperparameter optimization to the baselines. In this thesis, we aim to provide a fair evaluation of different encoding methods by using the same hyperparameter optimization budget for each method.

Our hyperparameter search space can be divided into two parts: one part contains the hyperparameters associated with the SAC learning algorithm, and the other part contains the hyperparameters associated with the encoding method ϕ. For each method, we sample a fixed number of hyperparameter combinations from the hyperparameter search space and use them to train our agent. This ensures that each method has the same opportunity to find hyperparameters that work well for the task. When the encoding method has more hyperparameters, the total hyperparameter search space becomes exponentially larger. This means we explore less of the search space associated with the SAC algorithm, potentially missing a good combination of hyperparameters for the problem.

Methods like VAEs require tuning several hyperparameters, such as the number of data points to train on, the learning rate, and the KL divergence weight β. In contrast, randomly initialized neural networks with a fixed architecture chosen *a priori* require no hyperparameter tuning. This allows us to allocate our entire hyperparameter optimization budget to the hyperparameters associated with SAC.

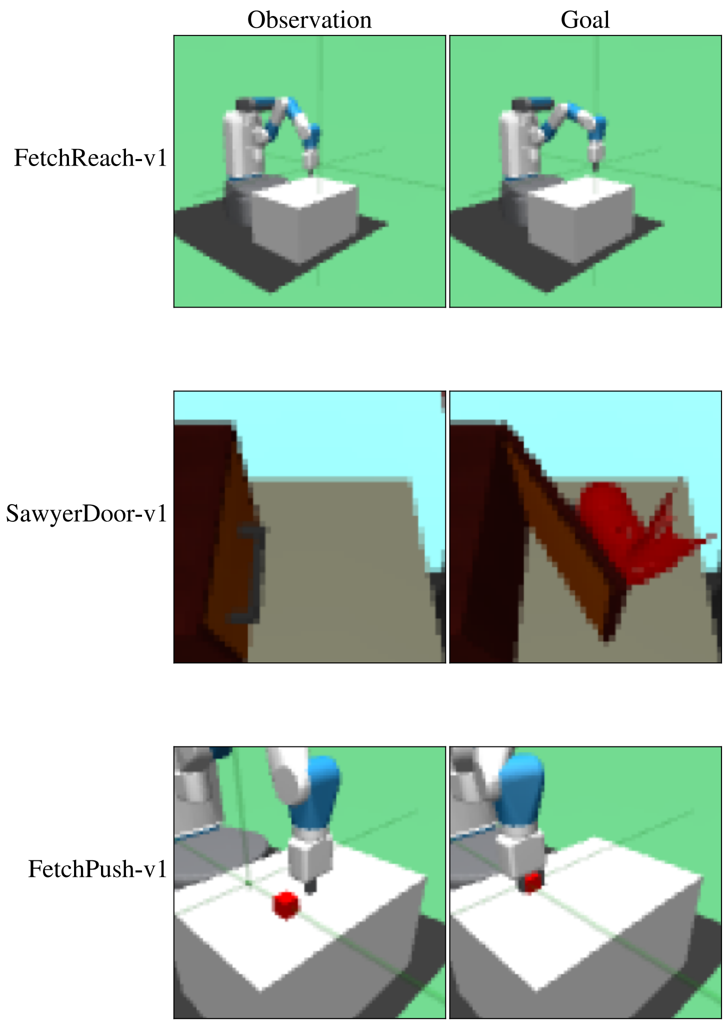

Figure 3.2: Initial image observations and desired goal images for each of the three simulated environments. The `SawyerDoor-v1` environment is a top-down view of a cupboard and robot arm.

Chapter 4

Results

4.1 `FetchReach-v1`

The `FetchReach-v1` environment is easy to solve, even directly using pixel distance. Given any encoding ϕ of image observations, we should have a maximum reward of 0 for $\phi(o) = \phi(o_g)$, which is satisfied when $o = o_g$. Thus, the agent can learn to solve the task by making the observation image match the goal image. Since the agent controls the robot, and the robot's movement is the only way that pixel values change in this environment, the agent can easily solve the task just by minimizing pixel distance. In fact, raw pixel distance is the most competitive method for this environment. The learning curves for each method are shown in figure 4.1. The variance in performance gives us an insight into the importance of hyperparameter selection. Figure 4.2 compares each method by averaging performance over all hyperparameters.

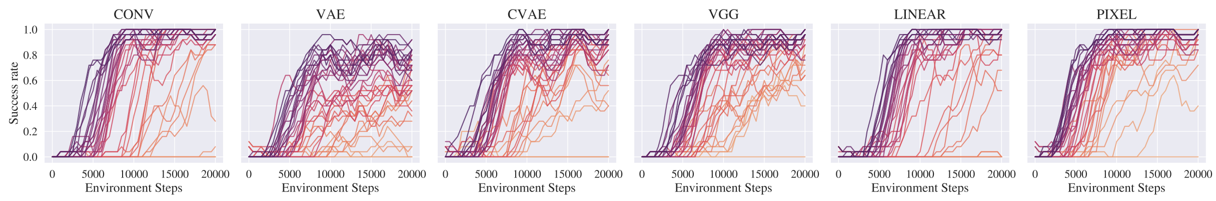

Figure 4.1: Learning curves for `FetchReach-v1`. Each point shows the success rate metric averaged over ten evaluation episodes for each time step. Each line represents the learning curve for a particular set of sampled hyperparameters. Lines are coloured according to the area under the learning curve, which indicates learning speed and overall performance.

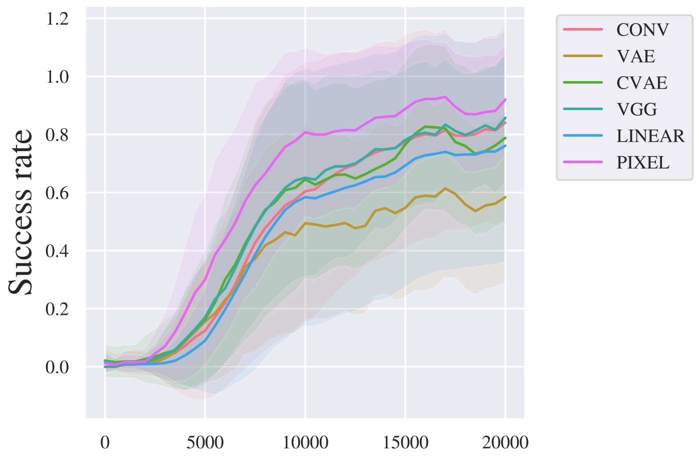

Figure 4.2: Combined learning curves for `FetchReach-v1`. Each line shows the the evaluation metric averaged over all sampled hyperparameters, with the area error bars corresponding to the standard deviation over all sampled hyperparameters.

While the VAE method has some hyperparameters that can perform competitively, we can see that over all sampled hyperparameters, many of them perform worse than the other baselines. We hypothesize that this is because a portion of VAE's hyperparameter optimization budget was dedicated to hyperparameters associated with training the VAE, such as β and the learning rate. Furthermore, the representation learned by the VAE may be comparatively poor, since the VAE is only trained on 1000 images collected via an agent with a randomly acting policy. These images may not represent the entire distribution of observation images, leading to nonsense latent vectors for images outside of the training distribution. In contrast, PIXEL and CONV do not rely on any learned representation, and are not susceptible to this problem.

Interestingly, we show that raw pixel distance is sufficient to solve this task directly from images, in contrast to notions suggested by prior work [20, 8, 28, 4, 21].

4.2 `SawyerDoor-v1`

The `SawyerDoor-v1` environment has two metrics of success: how close the door angle is to the goal door angle, and how close the robotic arm's end effector is to the goal position. No method achieves 100% success, which is likely due to the fact that small perturbations in state result in no discernible difference in pixel space. These perturbations can be the difference between success and failure according to the thresholds defined for the environment. We therefore additionally report the final distance to the goal for both the door and the robot arm, as shown in figure 4.4. This environment requires significantly more environment steps to run than `FetchReach-v1` (1,000,000 versus 20,000). It is also very slow to run in terms of wall clock time, because SAC does a parameter update at every time step. Therefore, we only sample 8 hyperparameters combinations from the hyperparameter search space for each method.

Hyperparameter choice is very important for this problem, with some choices of hyperparameters barely progressing past the initial performance, and others performing quite well. Even when averaged over all hyperparameter samples, CONV significantly outperforms other methods in terms of both success on the door angle and success on the robot position, as shown in figure 4.5. CONV performs almost one standard deviation better on average than the next-best method, CVAE. PIXEL and LINEAR methods perform similarly, and both perform much better than VAE. Interestingly, CVAE performed much better than VAE. Further investigation showed that the images sampled for VAE training by the randomly acting policy did not introduce enough variability to learn a good latent representation. The environment does not reset the door and robot arm positions when a new episode begins, and the robot arm does not manage to open the door in the 1000 time steps used to collect data for VAE training. See figure 4.3 for an example of poor reconstruction on this environment.

4.3 `FetchPush-v1`

We found that no method was able to solve this task using only images. To confirm the correctness of our implementation of SAC, we removed the convolutional encoder and used ground-truth states and goals as inputs to the agent. The agent learned to solve the task from states within 500,000 time steps.

Examining the learned behaviour from each method, the agent always learned to move the robot arm to the position shown in the goal image, and ignored the object. Once an agent learns this behaviour, it becomes trapped in a local optimum and fails to discover that moving the object can further increase the reward. We investigate the reasons for this behaviour further in section 6.1.

This is an interesting finding. Prior work has managed to solve this task directly from images using VAEs and CVAEs, but makes several additional modifications that we did not include in this work. Prior methods learn a policy directly on top of the latent representations learned by the VAE, whereas we learn the policy end-to-end directly from images. Prior methods also generate goals intrinsically by sampling from the Gaussian prior learned by the VAE and using these latent representations as goals, whereas we use the goals provided by the environment. This results in a kind of ablation study on prior work showing that rewards based on learned latent representations are not sufficient to solve the task directly from images.

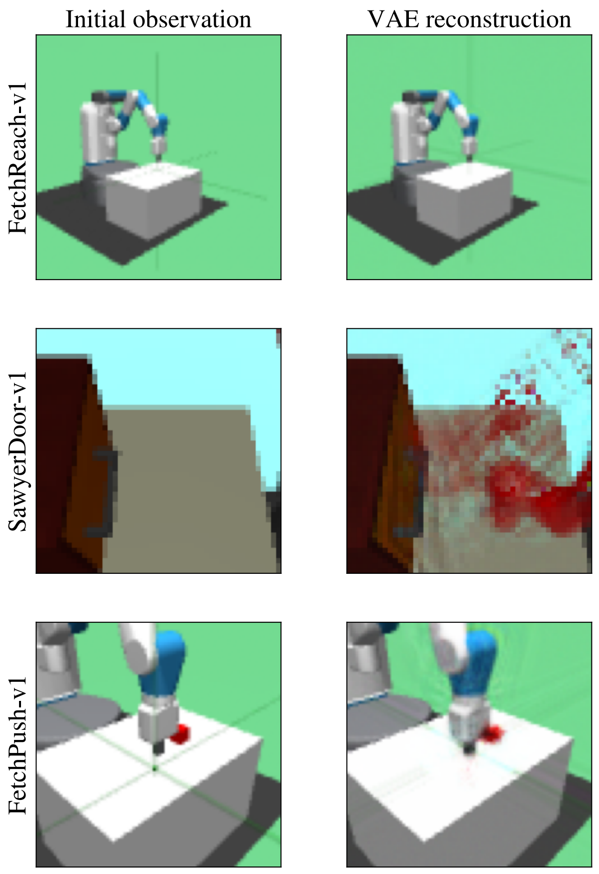

Figure 4.3: VAE reconstructions of initial observations. The data collected by the randomly acting policy for `SawyerDoor-v1` did not often include the initial state, so the initial state has poor reconstruction.

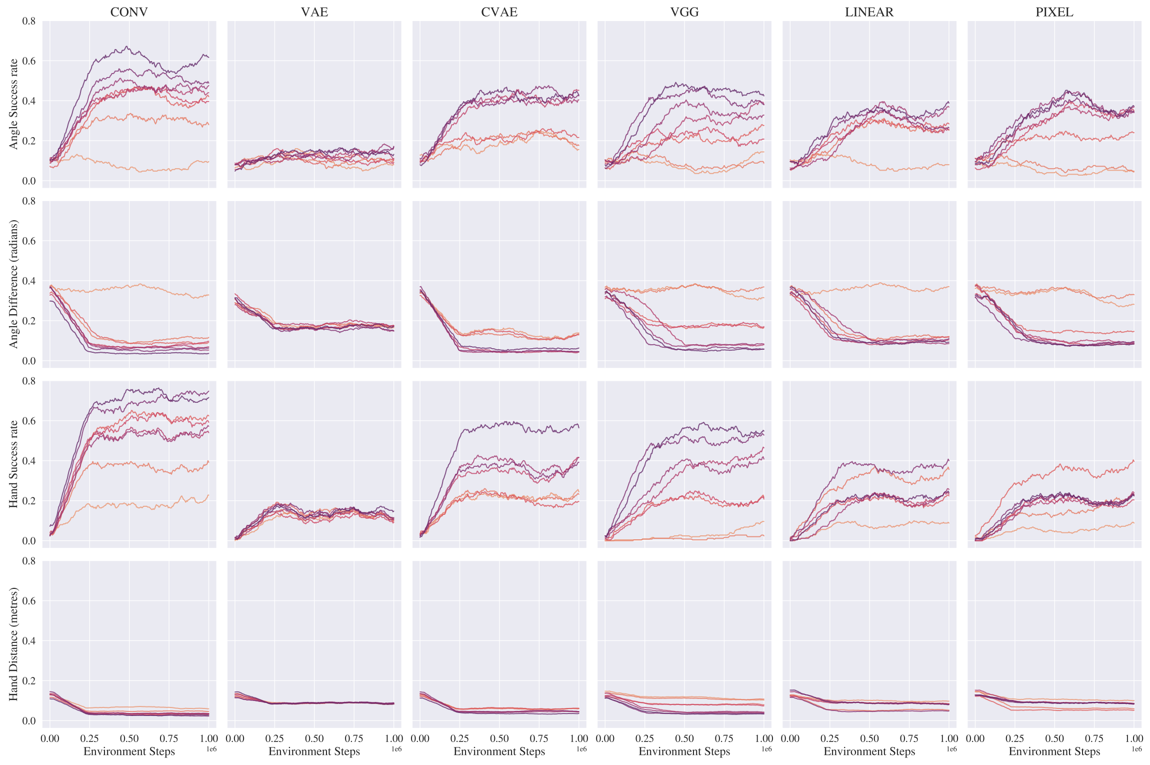

Figure 4.4: Learning curves for `SawyerDoor-v1`. Each point shows the evaluation metric averaged over ten evaluation episodes for each time step. Each line represents the learning curve for a particular set of sampled hyperparameters. Lines are coloured according to the area under the learning curve, which indicates learning speed and overall performance.

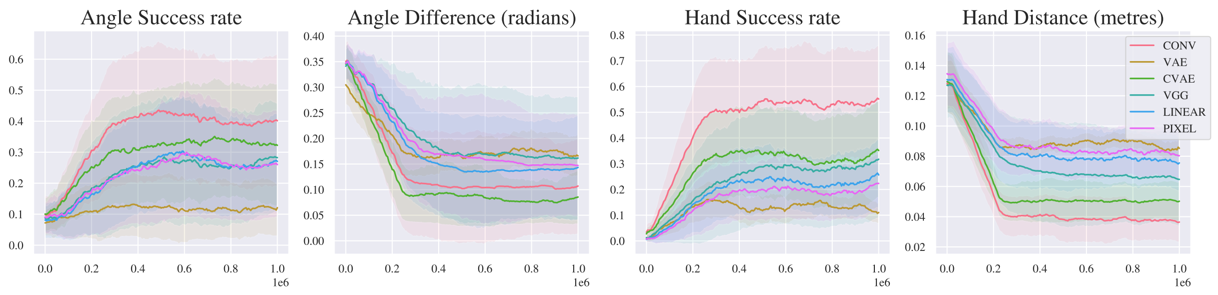

Figure 4.5: Combined learning curves for `SawyerDoor-v1`. Each line shows the the evaluation metric averaged over all sampled hyperparameters, with the area error bars corresponding to the standard deviation over all sampled hyperparameters.

Chapter 5

Discussion

5.1 Criterion for Success with Learning-Free Methods

In contrast to prior work, we show that simple methods that operate directly in pixel space like PIXEL, or that use learning-free feature extraction like CONV and LINEAR, are sufficient to solve `FetchReach-v1` and `SawyerDoor-v1`. What properties do these environments have that make them easy to solve? We propose the following criteria:

1. **Easy exploration task**: A task is an easy exploration task if a randomly acting agent is able to find the goal with some significant probability. These environments often have state spaces that can easily be controlled by the agent, so that the agent can quickly cover a large region of state space using only random actions.
2. **Matching reward gradients and latent reward gradients**: The gradient of the reward function with respect to the state vector points in approximately the same direction as the gradient of the latent reward function. This means that the agent can approximately optimize the reward function by optimizing the latent reward function.

To evaluate the first criterion, we task a randomly acting agent with exploring the environment until it either reaches the goal or reaches the 50 time step limit. For the `SawyerDoor-v1` task, we consider either door angle success or end-effector position success to be a success. We repeat this process for 100 episodes to estimate the proportion of successes a randomly acting agent would find. We find that a randomly acting agent finds goals in `SawyerDoor-v1` 36% of the time, in `FetchReach-v1` 14% of the time, and in `FetchPush-v1` 8% of the time. The agent does not have access to the underlying state information and therefore cannot know whether it has actually reached the goal. The purpose of this experiment is simply to verify that it possible to reach an arbitrary goal without any advanced exploration techniques.

To verify the second criterion, we need to compute two gradients: the gradient of the true reward function with respect to the state and the gradient of the latent reward function with respect to the state. We know that the true reward function is $r(s, a, s', g) = -\|s' - g\|_2$, and its gradient with respect to s' is $-(s' - g)/\|s' - g\|_2$. This gives us the direction in state space that most quickly maximizes the reward.

We want to compute the gradient of the latent reward function with respect to the state s', but must first pass through f_o the function that maps states to observations. This is a rendering operation that is nondifferentiable.[1] Therefore, we need to approximate the gradient of the latent reward function. We define our procedure for estimating this gradient in algorithm 1. We use r and ℓ to diffentiate between true reward and the latent reward respectively.[2]

Here we justify the gradient estimation procedure used in algorithm 1. Consider a multivariate function $f : \mathbb{R}^n \to \mathbb{R}$. The first-order Taylor series approximation of f around x is given by

$$f(x + h) \approx f(x) + h^\intercal \nabla_x f.$$

Rearranging we get,

$$f(x + h) - f(x) \approx h^\intercal \nabla_x f.$$

The term on the right hand side is the directional derivative of f at x along h. Define our approximation of $\nabla_x f$ to be δ. This gives

$$f(x + h) - f(x) \approx h^\intercal \delta.$$

We can collect a number of examples of h_i and $f(x + h_i)$ to generate a system of linear

[1]Rendering can be differentiable in principle, but capturing light from a camera in real life is not.

[2]The end-effector position of the robot is controlled using a mocap ("motion capture"), and we do positional control by setting the desired location of the mocap and performing inverse optimal control to plan a path from the current mocap position to the desired mocap position. Because of limitations in the design of the robot model as well as additional constraints on joint angles of the robot, it is not possible to exactly set the end-effector position to any arbitary desired location. This limitation is considered in the description of algorithm 1.

Algorithm 1 Approximate Gradient of Latent Reward Function

```
$s_0 \sim p(s_0)$                                            ▷ Sample an initial state
$g \sim \mathcal{G}$                                            ▷ Sample goal
$r \leftarrow -\|s - g\|_2$                                  ▷ Compute reward
$\ell \leftarrow -\|\phi(f_o(s)) - \phi(f_o(g))\|_2$          ▷ Compute latent reward
$\Delta_s \leftarrow \{\}$                                   ▷ To store changes in state
$\Delta_\ell \leftarrow \{\}$                                ▷ To store changes in latent reward
for $n \leftarrow 1 \dots N$ do:
    $h \sim S^{|\mathcal{S}|}$                               ▷ Sample a vector from the unit sphere in state space
    env.set_state($s + \alpha h$)                            ▷ Attempt to move mocap to new location
    $s' \leftarrow$ env.get_state()                          ▷ Compute actual new mocap position
    $\ell' \leftarrow -\|\phi(f_o(s')) - \phi(f_o(g))\|_2$    ▷ Compute new latent reward
    $\Delta_s \leftarrow \Delta_s \bigcup \{s' - s\}$        ▷ Store actual change in state
    $\Delta_\ell \leftarrow \Delta_\ell \bigcup \{\ell' - \ell\}$   ▷ Store change in latent reward
$\delta = \min_\delta \|\Delta_s \delta - \Delta_\ell\|_2^2$       ▷ Solve for $\delta$ by linear least-squares
return $\delta$
```

equations

$$\begin{bmatrix} h_0^\intercal \\ h_1^\intercal \\ \vdots \\ h_N^\intercal \end{bmatrix} \delta \approx \begin{bmatrix} f(x+h_0) - f(x) \\ f(x+h_1) - f(x) \\ \vdots \\ f(x+h_N) - f(x) \end{bmatrix}.$$

By solving for δ using least-squares regression we approximately recover the gradient $\nabla_x f$.

To validate algorithm 1, we compared the analytically derived gradient of the true reward function r to an estimate computed by algorithm 1. Using this algorithm, we can consistently approximate the gradient over large regions of state space in each environment. We compare the approximate gradient to the true gradient using cosine distance

$$d(u, v) = 1 - \frac{\langle u, v \rangle}{\|u\|_2 \|v\|_2},$$

which is the cosine of the angle between two equal-length vectors u and v, shifted to have a minimum of 0 when the vectors point in the same direction and to have a maximum of 2 when the vectors point in the opposite direction. The average cosine distance between the true gradient and the estimated gradient over each environment is less than 0.01 when N scales exponentially with the dimensionality of the state space $\mathcal{S}$.

We evaluate criterion 2 by computing the approximate gradient of the latent reward function over many points in state space for each environment, and compare it to the analytic gradient of the true reward function using the cosine distance. We denote by d the average cosine distance over all sampled points. The average cosine distances are summarized in table 5.1.

	CONV	CVAE	VAE	VGG	LINEAR	PIXEL
`FetchReach-v1`	0.26	0.25	0.42	0.31	0.28	0.23
`SawyerDoor-v1`	0.74	0.79	0.80	0.94	0.82	0.79
`FetchPush-v1`	0.68	0.66	0.77	0.71	0.73	0.67

Table 5.1: Average cosine distance d between true reward gradient and approximate latent gradient.

We find that for all environments and all encoders, d is less than 1, so the gradients agree at least somewhat. For `FetchReach-v1` they point very strongly in the same direction. We find that d is predictive of the performance of the corresponding encoding method on that environment. For example, PIXEL has the lowest cosine distance for `FetchReach-v1` and has the highest performance. Similarly, VAE has the highest cosine distance and has the lowest performance (see figure 4.2). For `SawyerDoor-v1`, CONV has the lowest cosine distance and the highest performance (see figure 4.5). The relationship is less clear for `FetchPush-v1` which was not solved by any method.

We also tested other potential criteria for success. The results of these experiments turned out to be insufficient to explain our results, or did not support the hypotheses we had formed. A selection of these experiments are detailed in the appendix.

Chapter 6

Conclusions

6.1 Limitations and Future Work

Learning-free encodings of images are very general, since they can be applied to any image-based scenario. Because of their generality, they are not specialized to work with particular environments. In contrast prior methods use specialized representations that are tuned to work with object manipulation tasks. Wulfmeier *et al.* [28] use tasks similar to the `FetchPush-v1` environment, but focus on object-centric representations that are well-suited to these tasks. Similarly, successors to RIG use learned representations that are conditioned on the initial observation [19] so that latent representations include information about the object colour. Unlike this prior work, our method did not succeed on the `FetchPush-v1` task. We hypothesize that two factors caused this failure.

First, the goals given to the agent are too difficult for it to immediately succeed directly from images. Methods like RIG and its successors, automatic goal generation methods like those presented by Florensa *et al.* [8] and representation methods explored by Wulfmeier *et al.* [28] gradually increase the difficulty of the goals presented to the agent. In contrast, we use the goals sampled directly from the evaluation distribution of goals provided by the environment, which may be initially too hard for the agent to solve. In fact, we found that an agent trained directly on the ground truth state using the evaluation goal distribution took up to 500,000 time steps to begin learning a successful policy.

The second reason for failure is because learning-free representations are more heavily influenced by the robot arm movement than the object movement, even though the object movement is the only part of the environment that determines success. We determined that this was the case by manually moving the object while keeping the robot arm fixed, and also the opposite, keeping the object fixed while moving the robot arm. We capture image observations of the environment in each of these scenarios and compute the dimension-wise variance in latent representations for different encoding methods. For learning-free representations, moving the robot arm results in much more variance in the latent representations than moving the object (see figure 6.1). In contrast, the variance in latent representations for VAE and CVAE are more balanced. Because the robot movement dominates the latent representation, it also dominates the computation of the latent distance to the goal, and so the agent minimizes the latent distance to the goal by greedily moving the robot to the position in the goal image.

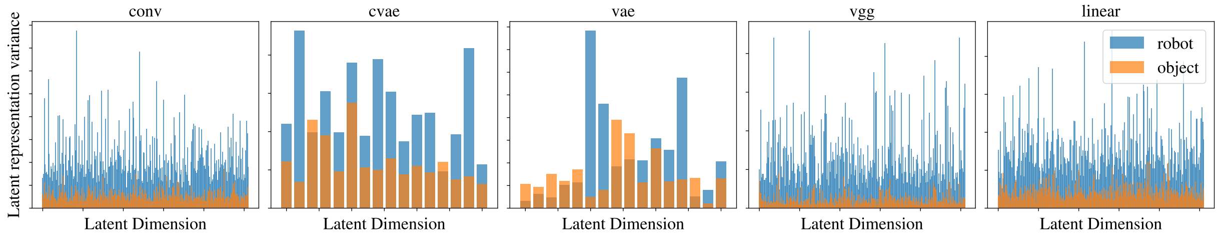

Figure 6.1: `FetchPush-v1` object versus latent variance. Shows the variance over each dimension of the latent representation when moving only the object versus moving only the robot arm.

Despite the VAE and CVAE latent representations being more balanced in terms of object and robot arm influence, they are still unable to learn the `FetchPush-v1` task. All agents just learn to move the robot arm to the position shown in the goal image. We therefore conclude that modifying the goal sampling distribution, as in other work, is critical for success in these environments. In future work, we propose incorporating methods for goal generation with these simple learning-free methods to determine the importance of learned representations.

We also did not test these methods on environments where consecutive time steps share very little overlapping pixels. In the robotics environments we tested, consecutive observations are quite similar, so raw pixel distance changes smoothly over time. This provides a nicely shaped reward for our agent. However, there are other goal-conditioned environments from images, like point mass navigation tasks, where pixel distance is essentially meaningless. Prior approaches to solving this kind of task involving learning a latent representation where distance between latent vectors corresponds to the amount of time a goal-conditioned policy would take to go from one to the other [7]. It is not clear whether a learning-free method would be able to solve this kind of task.

This thesis focuses on the capabilities of learning-free representations. However, there is a vast literature of methods for learned encodings of images that have largely not been considered for reinforcement learning. There are numerous dimensionality reduction techniques like principal components analysis (PCA) and multidimensional scaling (MDS) that can learn low-dimensional latent representations of images, while the literature seems largely constrained to using VAEs and CVAEs for this purpose. Furthermore, there are many self-supervised representation learning methods like momentum contrast (MoCo) [13] and SimCLR (A Simple Framework for Contrastive Learning of Visual Representations) [6] that have yet to be fully explored in reinforcement learning. In fact, CURL, which was the state-of-the-art for continuous control from images for some time, was only recently published in 2020, and used contrastive learning, a form of self-supervised learning, to help guide a good image representation for the policy and critic to learn from [15]. Expanding the way we transform images in RL beyond VAEs and basic convolutional neural networks could provide a rich avenue for future work.

6.2 Conclusion

In this thesis we train an RL agent to reach arbitrary goals specified as images. The agent only has access to image observations and the goal image, and must learn to match that goal image. We train an agent to maximize the reward $r = -\|\phi(o) - \phi(o_g)\|_2$ where ϕ is any method for transforming images into latent vectors. While prior work uses learned representations for ϕ, such as the encoder of a variational autoencoder, we show that certain problems can be solved using learning-free representations. In particular, we show that using raw pixel distance outperforms all encoding methods for `FetchReach-v1`. We also show that using features from a randomly initialized convolutional network can even outperform learned latent representations on harder tasks like `SawyerDoor-v1`. We demonstrate the importance of considering simple methods for goal-conditioned RL from images.

References

[1] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[2] Adrià Puigdomènech Badia, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskyi, Zhaohan Daniel Guo, and Charles Blundell. Agent57: Outperforming the Atari human benchmark. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 507–517. PMLR, 13–18 Jul 2020.

[3] Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. Journal of Artificial Intelligence Research 47, pages 253-279, 2012.

[4] Homanga Bharadhwaj, Animesh Garg, and Florian Shkurti. LEAF: Latent Exploration Along the Frontier. *arXiv e-prints*, page arXiv:2005.10934, May 2020.

[5] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. In *International Conference on Learning Representations*, 2019.

[6] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 1597–1607. PMLR, 13–18 Jul 2020.

[7] Carlos Florensa, Jonas Degrave, Nicolas Heess, Jost Tobias Springenberg, and Martin Riedmiller. Self-supervised learning of image embedding for continuous control, 2019.

[8] Carlos Florensa, David Held, Xinyang Geng, and Pieter Abbeel. Automatic goal generation for reinforcement learning agents. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1515–1528, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[9] Dibya Ghosh, Abhishek Gupta, and Sergey Levine. Learning actionable representations with goal-conditioned policies, 2018.

[10] David Ha and Jürgen Schmidhuber. World models, 2018.

[11] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. 2020.

[12] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2555–2565. PMLR, 09–15 Jun 2019.

[13] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick. Momentum contrast for unsupervised visual representation learning. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9726–9735, 2020.

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

[15] Michael Laskin, Aravind Srinivas, and Pieter Abbeel. CURL: Contrastive unsupervised representations for reinforcement learning. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 5639–5650. PMLR, 13–18 Jul 2020.

[16] Misha Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. Reinforcement learning with augmented data. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 19884–19895. Curran Associates, Inc., 2020.

[17] V. Lempitsky, A. Vedaldi, and D. Ulyanov. Deep image prior. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9446–9454, 2018.

[18] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.

[19] Ashvin Nair, Shikhar Bahl, Alexander Khazatsky, Vitchyr H. Pong, G. Berseth, and S. Levine. Contextual imagined goals for self-supervised robotic learning. In *CoRL*, 2019.

[20] Ashvin V Nair, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine. Visual reinforcement learning with imagined goals. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[21] Suraj Nair, Silvio Savarese, and Chelsea Finn. Goal-aware prediction: Learning to model what matters. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 7207–7219. PMLR, 13–18 Jul 2020.

[22] OpenAI, Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning. 2019.

[23] Matthias Plappert, Marcin Andrychowicz, Alex Ray, Bob McGrew, Bowen Baker, Glenn Powell, Jonas Schneider, Josh Tobin, Maciek Chociej, Peter Welinder, Vikash Kumar, and Wojciech Zaremba. Multi-goal reinforcement learning: Challenging robotics environments and request for research, 2018.

[24] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1312–1320, Lille, France, 07–09 Jul 2015. PMLR.

[25] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, October 2017.

[26] Yuval Tassa, Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, and Nicolas Heess. dm_control: Software and tasks for continuous control, 2020.

[27] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *IROS*, pages 5026–5033. IEEE, 2012.

[28] Markus Wulfmeier, Arunkumar Byravan, Tim Hertweck, I. Higgins, A. Gupta, T. Kulkarni, M. Reynolds, Denis Teplyashin, Roland Hafner, T. Lampe, and Martin A. Riedmiller. Representation matters: Improving perception and exploration for robotics. *ArXiv*, abs/2011.01758, 2020.

[29] Richard Zhang, Phillip Isola, Alexei A. Efros, E. Shechtman, and O. Wang. The unreasonable effectiveness of deep features as a perceptual metric. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 586–595, 2018.

Appendix A

APPENDICES

A.1 Hyperparameter Search Space

Table A.1 shows all hyperparameters for our experiments, as well as their possible values. When we say we sampled n hyperparameter combinations, we mean that, for each hyperparameter that can take on multiple values, we samples a single value for that hyperparameter.

Latent dimensions of ϕ	$\{128, 256\}$ or $\{8, 16\}$ for VAE/CVAE
Hidden layer sizes of actor/critic networks	$\{[256, 256], [512, 512], [1024, 1024]\}$
Discount factor γ	0.95
Initial temperature for SAC	$\{0.1, 1.\}$
Learning rate	$\{0.001, 0.0005, 0.0001\}$
Actor update frequency	$\{1, 2\}$
Critic τ	$\{0.005, 0.001\}$
Batch size	256
Gradient steps per environment step	2
Replay buffer capacity	$1,000,000$

Table A.1: Hyperparameters

A.2 Additional Experiments

We were interested to know why certain methods performed better than others on various environments. Here we briefly describe a selection of hypotheses and experiments, along with what the results of those experiments show.

A.2.1 Correlation between Latent Reward and True Reward

We compute the true reward as $r(s, a, s', g) = -\|s' - g\|_2$, using the underlying states and goal state from the simulator. For each environment and encoding method, we can place the agent in a random point in state space with a goal from the evaluation distribution, and we can then compute both the true and latent rewards. Repeating this for many points in state space allows us to determine how well latent reward corresponds with true reward. See figure A.1.

We can see that for the `FetchReach-v1` environment, we have strong correlation between latent and true reward for the PIXEL and LINEAR methods, and somewhat strong correlation for the CONV and VGG methods. However, we don't see strong correlation for the VAE and CVAE methods. Interestingly, for the `SawyerDoor-v1` environment, we see the strongest correlation for the VGG method. There is almost no correlation for any method for the `FetchPush-v1` environment (in fact, a least-squares regression predicts a negative relationship between the two rewards).

A.2.2 Preservation of State Space Topology in Latent Space

We wondered if the topology of the state space is somehow preserved in the latent space. This means that the mapping $\mathcal{S} \to \mathcal{O} \to \mathcal{Z}$ from state space to observation space (rendering) and from observation space to latent space somehow preserves the connectedness and distances between points.

To visualize this theory, we consider a 2D subset of state space for each environment. For example, in the `SawyerDoor-v1` environment, this 2D subset corresponds to the changes in the door angle θ and the horizontal position of the robot arm x. We can manually set the environment to points on a uniform grid over this 2D subspace. For each point in this grid, we can capture an image and compute the latent representation. This gives us a set of points in state space and their corresponding representations in latent space. We can then find a 2D manifold inside the set of latent representations that preserves geodesic distances

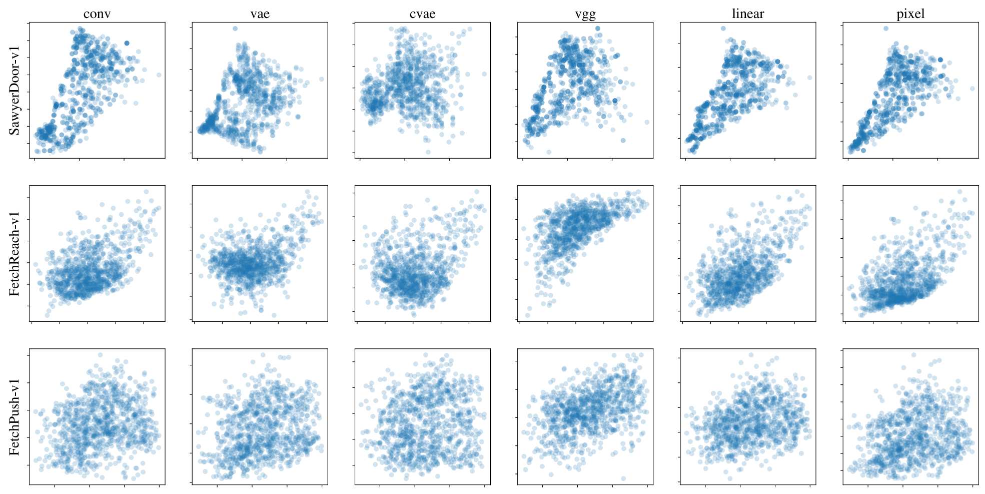

Figure A.1: Correlation between true reward and latent reward. True reward on x axis, latent reward on y axis.

using the isomap algorithm. Finally, we can project our set of latent representations onto the low-dimensional 2D manifold learned by isomap, giving us pairs of 2D coordinates: one in state space and one in the projected latent space. We can then compare the topology of these two sets of points by colouring each point according to its coordinates, with one coordinate for hue and one coordinate for value.

In figure A.2 we choose the (x, y) plane with z fixed in midair to be the subspace of state space to map. We can see that the PIXEL, LINEAR and CONV methods all have similar latent space topologies. VGG has a similar topology to the state space, but is more distorted than PIXEL, LINEAR, and CONV. Finally, VAE and CVAE are the most distorted, which is to be expected, since their latent spaces are not learned to be "flat" but rather to form a diagonal Gaussian distribution. There is no obvious mapping from an n-dimensional ball to a flat 2D surface.

In figure A.3, we choose the door angle θ and the horizontal position of the robot's end-effector x to be our 2D subspace of state space. Interestingly, PIXEL has the best topology preservation, followed by LINEAR. Despite CONV performing best here, its latent space seems to be more distorted than PIXEL and LINEAR. There seems to be no clear relationship between the results of this experiment and the performance of different

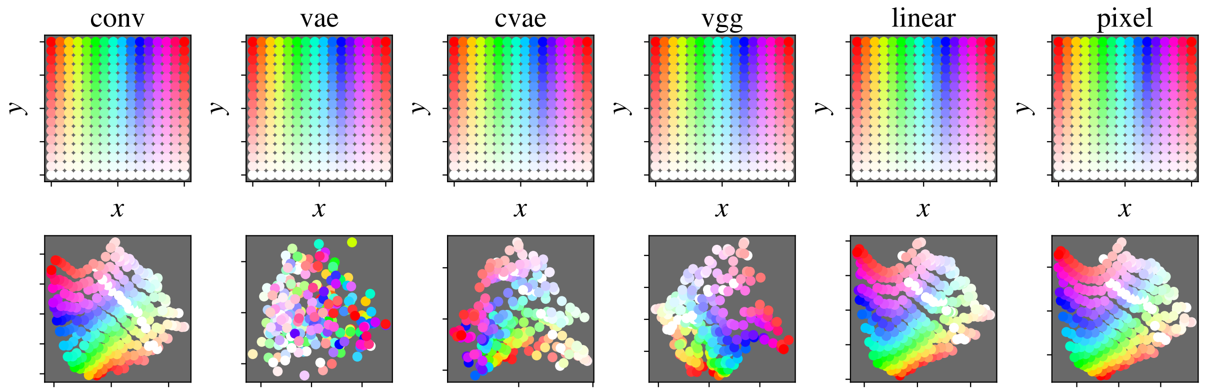

Figure A.2: `FetchReach-v1` latent space visualization.

encoding methods.

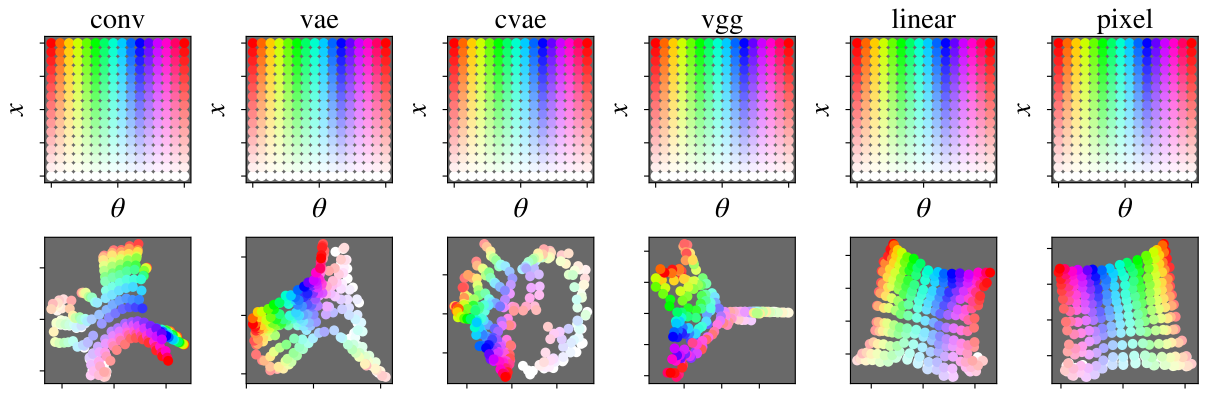

Figure A.3: `SawyerDoor-v1` latent space visualization.

Finally, in figure A.4, we choose our 2D subspace to be that with the robot arm and object occupying the same (x, y) coordinates, restricting those coordinates to the surface of the table. We see that PIXEL and CONV encode state space similarly. Since the robot arm is moved in conjunction with the object, we can expect these encodings to be somewhat similar to those for `FetchReach-v1`. Interestingly, the VAE and CVAE spaces seem to be better represented here than in `FetchReach-v1`, perhaps because more of latent space is dedicated to object position. Again, it is not clear that latent space topology relates to final performance of different encoding methods.

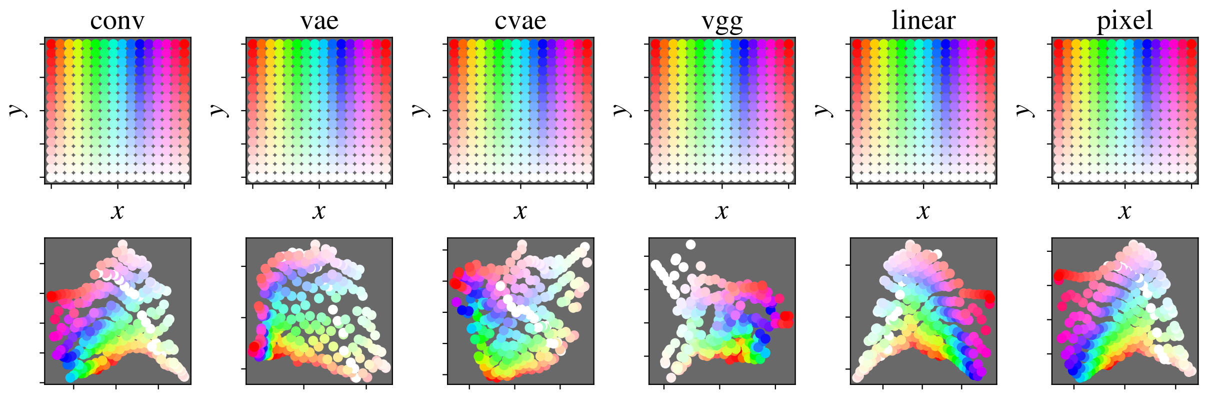

Figure A.4: `FetchPush-v1` latent space visualization.