# Representation Learning for Perception and Control

*Aravind Srinivas Lakshminarayanan*

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2021-197
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2021/EECS-2021-197.html

August 13, 2021

# Representation Learning for Perception and Control

by

Aravind Srinivas Lakshminarayanan

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Pieter Abbeel, Chair
Professor Ken Goldberg
Professor Trevor Darrell

Summer 2021

Representation Learning for Perception and Control

Abstract

Representation Learning for Perception and Control

by

Aravind Srinivas Lakshminarayanan

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Pieter Abbeel, Chair

The goal of extracting reusable and rich representations that capture *what you care about* for downstream tasks remains challenging even though the field of deep learning has made tremendous progress in this direction. This thesis presents a few promising contributions to further that goal. The two axes of contributions are: (1) self-supervised (or unsupervised) representation learning; (2) deep neural network architectures powered by self-attention. Progress in architectures and the ability to leverage massive amounts of unlabeled data have been responsible for major advances in NLP such as GPT-x and BERT. This thesis presents small steps towards realizing such progress for perceptual and reinforcement learning tasks. This is a thesis by articles containing four articles, two focused on computer vision benchmarks, with the other two focused on reinforcement learning.

With respect to the first axis, the thesis presents three articles: (1) Data-Efficient Image Recognition using Contrastive Predictive Coding (CPCv2); (2) Contrastive Unsupervised Representations for Reinforcement Learning (CURL); (3) Reinforcement Learning with Augmented Data (RAD). The first two articles explore a form of unsupervised learning called contrastive learning, a technique better suited for raw inputs such as images compared to generative pre-training that is popular for language. The first article presents results for label-efficient image recognition. The second article presents the benefits of contrastive learning for sample-efficient reinforcement learning from pixels. Contrastive learning in practice is heavily dependent on data augmentations, and the third article presents a detailed investigation and discussion of its role.

As for the second axis, the thesis presents a thorough empirical investigation of the benefits of self-attention and Transformer-like architectures for computer vision through the article: Bottleneck Transformers for Visual Recognition. Self-attention has revolutionized language processing but computer vision presents a challenge to vanilla Transformers through high resolution inputs that challenge the quadratic memory and computational complexity of the primitive. The article presents the empirical effectiveness of a straightforward hybrid composed of convolutions and self-attention and unifies the ResNet and Transformer based architecture design for computer vision.

To my mom

# Acknowledgments

I am thankful to my advisor Pieter Abbeel for supporting and mentoring me. He has given me the freedom to pursue my research ideas and facilitated my career growth immensely. Several times, I was taken seriously in conversations simply because I was his student, and being his student has undoubtedly opened up several fantastic career opportunities for me. Our weekly meetings were something I looked forward to for the optimism and ambition he espouses. Pieter inspires me to be meticulous, driven, and ambitious for several years down the line, and I am a big fan of his entrepreneurial drive and insights. Becoming his student has provided me a multi-dimensional outlook, even beyond academic research, that has significantly improved my understanding of the world. Thanks to John Canny for admitting me to Berkeley, Pieter for being willing to rotate with me, and Berkeley EECS, BAIR, and Trevor Darrell for the first-year fellowship. Trevor has created a fantastic and thriving ecosystem for AI Ph.D. students in Berkeley. In addition, I thank Trevor Darrell and Ken Goldberg for graciously agreeing to be on my committee and providing me encouraging feedback along the way. I also had the pleasure of interacting with other excellent faculty at Berkeley, including John Canny, who gave me the idea to work on contrastive learning for robotics way back in 2017; Alexei Efros, who is inspiring with his authenticity and taste, Angjoo Kanazawa for serving on my qualifying exam committee, and James Demmel, a legend in applied numerical linear algebra. I enjoyed explaining to him how Transformers work.

Thanks to John Schulman for working with me during my first year and hosting me as an OpenAI intern in 2018. John is still Pieter's best student undoubtedly and inspired me throughout my Ph.D. He certainly has shaped a lot of my thinking on what impactful research is. I am thankful to him for the trust and time he invested in me. The OpenAI internship profoundly impacted me, watching several famous researchers in action while just being in my first year of Ph.D. Ilya Sutskever, a hero for me, inspired me to study and do research on unsupervised learning. I thank Peter Chen and Jonathan Ho for working with me on Flow++ while I was new to generative modeling and large-scale deep learning. Their mentorship helped me a great lot.

Thanks to my friend Sherjil Ozair who pointed me to Aaron van den Oord's paper on Contrastive Predictive Coding (CPC). I still remember spending the entire night in Sweden while at ICML 2018 reading the CPC paper and bumping into Aaron himself the next day at the conference, excitedly asking him many questions on it. Big thanks to Aaron and Oriol Vinyals for hosting me as an intern at DeepMind in their team the following summer. The summer of 2019 that we spent scaling and improving CPC and trying various variants have been critical to my growth as a Deep Learning researcher. Thanks to my co-authors on CPCv2 - Olivier Henaff, Jeffrey De Fauw, Carl Doersch, Ali Razavi, Ali Eslami, and Aaron. During my internship, I also had great times with several folks at DeepMind, including Igor Babuschkin, Jeff Donahue, Karen Simonyan, Sander Dieleman, Yazhe Li, and Sherjil Ozair Roman Ring, Ankush Gupta, Tejas Kulkarni, Vlad Mnih, Max Jaderberg, Jacob Menick, Aaron and Oriol. Meeting and interacting with Alex Graves was a highlight. I also thank Nal Kalchbrenner and Tejas Kulkarni for several inspiring conversations. Both of them are inspiring researchers with unique tastes and views on deep learning architectures and generative modeling.

Thanks to Misha Laskin for working with me on CURL and RAD. I still remember having the 1:1 conversation with Pieter where we spotted an opportunity to apply contrastive learning and data augmentations to significantly simplify and speed up RL from pixels. Later, I wrote design docs about it and began the efforts with Misha. Thanks to Aaron for sowing the seeds for applying contrastive learning to improve the sample efficiency in RL.

I thank Ashish Vaswani for being my collaborator on Bottleneck Transformers, where I got to learn a lot from him on how to design self-attention architectures. It helped me build different thinking about deep learning architectures - considering compute efficiency, throughput and memory tradeoffs, and how accelerators could make more efficient use of the FLOPs in a model. I also thank Quoc Le for hosting me as an intern at Google Brain and pushing for the importance of data augmentations. I enjoyed talking to folks at Google Brain, including Ashish, Quoc, Ben Poole, Niki Parmar, Noam Shazeer, David Luan, Durk Kingma, Nal Kalchbrenner, Jonathan Ho, Colin Raffel, Tsung-Yi Lin, and Jeff Dean.

Thanks to all my labmates who have worked with me on this thesis: Misha Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto. I also had the chance to collaborate and/or write papers outside of this thesis with several others, including Jonathan Ho, Peter Xi Chen, Rocky Duan, Wilson Yan, Yunzhi Zhang, Lili Chen, Mandi Zhao, Ajay Jain, Sean'O Brien, Jeffrey Tao, Aaron Putterman, Allan Jabri, Chelsea Finn. Thanks to Allan Jabri, Michael Chang, Roshan Rao, Ashvin Nair, Haozhi Qi for many enjoyable conversations as Ph.D. students from the same cohort. Thanks to other Ph.D. students in Pieter's lab, including Olivia Watkins, Yuqing Du, Hao Liu, Josh Achiam, Josh Tobin, Jason Peng, Roshan Rao. Thanks to the barista at Philz Coffee, Brewed Awakening, Nefeli's, Asha Tea House, Caffeinated Coffee Company (beneficial during pandemic lockdowns), and Victory Point.

Thanks to Ilya Sutskever for profoundly changing my life at so many levels through his mentorship. A lot of the work done in this thesis contains elements he stands for: minimal innovation for maximum results, scale, compute. Ilya has maintained contact with me ever since my 2018 internship and has given a lot of invaluable mentorship along the way. I am also grateful to Andrej Karpathy for sharing his insights and research ideas during my conversations.

Thanks to my friends, in no particular order: Vinod Ganesan, Ilija Radosavovic, Tejas Kulkarni, Ashish Vaswani, Peter Chen, Nal Kalchbrenner, Sherjil Ozair, Ankur Handa, Ankush Gupta, Allan Jabri, Michael Chang. The initial few weeks of the lockdown were tough, and I thank Ilija for being my buddy for lunch or dinner almost every day. I am glad to have met him through the Berkeley AI lab, with an initial topic being computer vision benchmarks and Detectron, but broadening to pretty much everything later on. Ilija helped me by answering countless questions about benchmarks such as COCO, codebases like Detectron, and backbone architectures in general. I also thank my family and relatives for believing and supporting me.

I dedicate this thesis to my mom, Jayanthi, for all the hardship she has gone through to provide me with good education and make my life what it is. My mom inculcated in me values such as excellence, grit, and hard work; and has motivated me to go to the US for education and a successful career while she made the sacrifices, and I am forever indebted to her. Thank you, Amma.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

This chapter gives a brief overview of the thesis. It does not introduce the basics of machine learning such as supervised, unsupervised, reinforcement, deep learning, etc. From that perspective, the thesis is not self-contained and assumes familiarity on the reader's part, with foundations and basics of machine learning and deep learning. The reader is encouraged to check out Ian Goodfellow's book on Deep Learning [62] and Rich Sutton's book on Reinforcement Learning [172] if interested. Further, unsupervised learning, contrastive learning, and energy-based models are well covered in a tutorial by Yann LeCun [112]. The individual articles in the thesis are presented as chapters. Each chapter gives the motivation and basics of their techniques, such as contrastive learning, data augmentations, self-attention, etc. Considering the empirical nature of the thesis and the fact that deep learning is, in general, a practical discipline, the thesis has very little mathematical treatment of the associated techniques and only uses mathematical notations when necessary.

Deep learning of representations and its applications to computer vision and robotics/reinforcement learning from pixels is a topic of enormous interest and significance in the machine learning and artificial intelligence communities. It is seen as an important milestone and an essential ingredient necessary to progress towards the broader goal of general artificial intelligence and general-purpose machines and robots that automate a lot of human activity.

Since AlexNet [104] was introduced by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, the field has made tremendous progress in learning reusable and rich representations of high dimensional inputs through large deep neural networks. In computer vision, such representations have predominantly been achieved through supervised learning on large, carefully annotated datasets such as ImageNet [150] and COCO [121]. These datasets have led to influential backbone architectures, object detectors, segmentation models, and image captioning pipelines. On the other hand, the field of reinforcement learning (RL) has seen incredible progress by fusing established RL algorithms with deep neural networks as function approximators since DQN from DeepMind [132]. The subsequent progress since AlexNet and DQN for both supervised and reinforcement learning has been nothing short of outstanding. Naturally, the AI/DL community has been gravitating towards the next challenge and new techniques for learning even better representations. Could we rely on fewer labels, scale better with more data, design models faster for the same accuracy, learn quickly in an environment, etc.? These are the kind of questions that motivate a lot of the current

research in the community.

A simple guideline to making further progress in deep learning is to answer the question: How could I learn better representations of data than what is possible already? Progress in representation learning has usually been achieved through better learning objectives and better deep learning architectures. These are the two core axes of this thesis. For better learning objectives, the thesis studies a form of unsupervised learning called contrastive learning. It shows its benefits for label-efficient image recognition and sample-efficient reinforcement learning from pixels. For better deep learning architectures, the thesis studies the design of backbone architectures for object recognition tasks like classification, detection, and instance segmentation. Notably, it presents a straightforward yet effective way of combining convolutions and self-attention with a unification of the ResNet [73] and Transformer [186] architectures.

Let me elaborate on the motivations for each of the above.

Why should we care about better learning objectives and unsupervised learning? There are both philosophical and practical arguments for this. Turing Award Winner Yann LeCun has a famous quote on intelligence:

> *If intelligence is a cake, unsupervised learning is the cake; supervised learning is the icing on the cake; reinforcement learning is the cherry on the cake.*

This became quite popular in the community and is often referred to as LeCake. The argument Yann LeCun makes is simple: Unsupervised Learning presents orders of magnitude more bits for our brains to learn than Supervised and Reinforcement Learning and must guide the learning of bulk of the parameters in an intelligent system. A practical argument favoring unsupervised learning is: We have way more unlabeled data available than the amount of labeled data. Can we somehow take advantage of the abundantly available unlabeled data to improve all the models that work pretty well already for labeled tasks? If we do, could such a system be a lot more label-efficient and save us time in annotating large datasets? Further, could the same techniques that help us become label-efficient for passive intelligence like visual recognition or language understanding be more sample-efficient for active intelligence such as controlling locomotion robots or game-playing agents?

Chapter 2 presents the article "Data-Efficient Image Recognition using Contrastive Predictive Coding" [79], published in ICML 2020. This paper presented first-of-its-kind results at its publication by pushing the label-efficiency on the ImageNet benchmark up to 5x and showing improvements on recognition tasks across all data-regimes through contrastive pre-training. While the field is quite fast, and follow-up work such as SimCLR [26] and BYOL [64] have improved upon the architectures and implementation in this article, the article was the first to show the benefits of contrastive pre-training for label-efficient supervised image recognition on the ImageNet benchmark.

Chapter 3 and 4 present the articles "Contrastive Unsupervised Representations for Reinforcement Learning (CURL)" [165] (ICML 2020) and "Reinforcement Learning with Augmented Data (RAD)" [110] (NeurIPS 2020). CURL presented results for the first time that pixel-based reinforcement learning could be as sample-efficient as state-based. To clarify why this is relevant, continuous

control tasks often involve a complex sequence of actions to keep a robot satisfy a certain reward function, for example, walker robot walking, cheetah robot running, etc. While good off-policy RL algorithms existed to have fast learning from engineered state features (such as joint angles, velocities, etc.), a fast-enough method to learn directly from pixels was lacking. The core idea is that: If we could learn good features from pixels through a good representation learning algorithm, it should allow for as fast learning as state engineering. CURL established new state-of-the-art (for its time) on DMControl benchmarks, beating several existing relatively more complex baselines that learn world models in pixel or latent spaces. It also introduced the idea of using data augmentations and instance-based contrastive learning for RL, an approach popularly referred to in computer vision as SimCLR [26], a paper published by Geoffrey Hinton in the same conference, ICML 2020. Data augmentations played an important role in the strong results established in CURL. To study their importance agnostic of the contrastive learning objective, Chapter 4 discusses the RAD article (NeurIPS 2020). For several DMControl tasks, directly augmenting the input provides good or better performance relative to CURL. Nevertheless, both these ideas are relevant and useful in their form. The community has since developed better pipelines, that combine useful elements from both CURL and RAD, for better results on Atari, DMControl, and real-world robotic control. The reader is also highly encouraged to check out the results in CURL that use a detached encoder: i.e., no backpropagation of RL gradients to the convolutional encoder that learns representations. Such a recipe might likely be how we perform representation learning for RL in the future, and the early signs are promising.

Now, let me elaborate on the other axis of the thesis: Why should we care about better deep learning architectures? Major progress in deep learning has often come through better architectures: AlexNet, VGG, ResNet, LSTMs, WaveNet, Transformers, so forth. A noticeable trend is how people have always gone for universal primitives in the architectures that work across multiple problems (modalities). For example, sequence models in NLP were initially RNN (LSTM) based [169], but later evolved to convolution-based architectures [97] and then into pure attention models since the Transformer [186]. There is now a lot of interest in the community to make Transformer based models that use self-attention, work well for vision (either convolution-free or using hybrid models involving both convolutions and self-attention). Achieving a universality in the architecture pipelines across multiple modalities could be crucial to performing multimodal representation learning with ease at scale. Chapter 5 presents the article "Bottleneck Transformers for Visual Recognition (BoTNets)," published in CVPR 2021 to that end. It presented (concurrent to the more popular Vision Transformer (ViT) [43]) (then) state-of-the-art results with a straightforward hybrid design of convolutions and self-attention, unifying the ResNet and Transformer based architecture designs for computer vision recognition tasks. Since then, several works have been published in the field, improving the results presented in this article. Nevertheless, it has become clear that for efficient models that achieve good accuracy, one needs to have elements of convolutions even in attention-based architectures such as locality, early convolutions (for the stem), and multi-scale processing of which are present in the hybrid design of BoTNet.

# Chapter 2

# Image Recognition using Contrastive Learning

## 2.1 Introduction

Deep neural networks excel at perceptual tasks when labeled data are abundant, yet their performance degrades substantially when provided with limited supervision (Fig. 4.2, red). In contrast, humans and animals can learn about new classes of images from a small number of examples [108, 126]. What accounts for this monumental difference in data-efficiency between biological and machine vision? While highly structured representations (e.g. as proposed by [106]) may improve data-efficiency, it remains unclear how to program explicit structures that capture the enormous

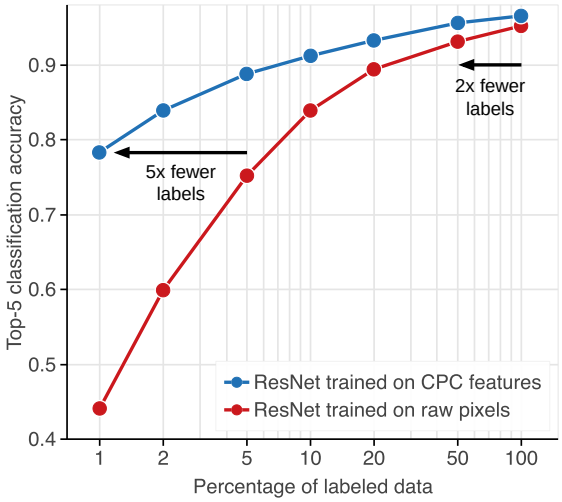



Figure 2.1: Data-efficient image recognition with Contrastive Predictive Coding. With decreasing amounts of labeled data, supervised networks trained on pixels fail to generalize (red). When trained on unsupervised representations learned with CPC, these networks retain a much higher accuracy in this low-data regime (blue).

complexity of real-world visual scenes, such as those present in the ImageNet dataset [150]. An alternative hypothesis has therefore proposed that intelligent systems need not be structured *a priori*, but can instead learn about the structure of the world in an unsupervised manner [10, 83, 111]. Choosing an appropriate training objective is an open problem, but a potential guiding principle is that useful representations should make the variability in natural signals more predictable [181, 194, 148]. Indeed, human perceptual representations have been shown to linearize (or 'straighten') the temporal transformations found in natural videos, a property lacking from current supervised image recognition models [78], and theories of both spatial and temporal predictability have succeeded in describing properties of early visual areas [146, 136]. In this work, we hypothesize that spatially predictable representations may allow artificial systems to benefit from human-like data-efficiency.

Contrastive Predictive Coding (CPC, [135]) is an unsupervised objective which learns predictable representations. CPC is a general technique that only requires in its definition that observations be ordered along e.g. temporal or spatial dimensions, and as such has been applied to a variety of different modalities including speech, natural language and images. This generality, combined with the strong performance of its representations in downstream linear classification tasks, makes CPC a promising candidate for investigating the efficacy of predictable representations for data-efficient image recognition.

Our work makes the following contributions:

- We revisit CPC in terms of its architecture and training methodology, and arrive at a new implementation with a dramatically-improved ability to linearly separate image classes (from 48.7% to 71.5% Top-1 ImageNet classification accuracy, a 23% absolute improvement), setting a new state-of-the-art.

- We then train deep neural networks on top of the resulting CPC representations using very few labeled images (e.g. 1% of the ImageNet dataset), and demonstrate test-time classification accuracy far above networks trained on raw pixels (78% Top-5 accuracy, a 34% absolute improvement), outperforming all other semi-supervised learning methods (+20% Top-5 accuracy over the previous state-of-the-art [205]). This gain in accuracy allows our classifier to surpass supervised ones trained with $5\times$ more labels.

- Surprisingly, this representation also surpasses supervised ResNets when given the entire ImageNet dataset (+3.2% Top-1 accuracy). Alternatively, our classifier is able to match fully-supervised ones while only using half of the labels.

- Finally, we assess the generality of CPC representations by transferring them to a new task and dataset: object detection on PASCAL VOC 2007. Consistent with the results from the previous sections, we find CPC to give state-of-the-art performance in this setting (76.6% mAP), surpassing the performance of supervised pre-training (+2% absolute improvement).

## 2.2 Experimental Setup

We first review the CPC architecture and learning objective in section 2.2, before detailing how we use its resulting representations for image recognition tasks in section 2.2.

### Contrastive Predictive Coding

Contrastive Predictive Coding as formulated in [135] learns representations by training neural networks to predict the representations of future observations from those of past ones. When applied to images, CPC operates by predicting the representations of patches below a certain position from those above it (Fig. 2.2, left). These predictions are evaluated using a contrastive loss [29, 68], in which the network must correctly classify 'future' representations among a set of unrelated 'negative' representations. This avoids trivial solutions such as representing all patches with a constant vector, as would be the case with a mean squared error loss.

In the CPC architecture, each input image is first divided into a grid of overlapping patches $\boldsymbol{x}_{i,j}$, where $i, j$ denote the location of the patch. Each patch is encoded with a neural network $f_\theta$ into a single vector $\boldsymbol{z}_{i,j} = f_\theta(\boldsymbol{x}_{i,j})$. To make predictions, a masked convolutional network $g_\phi$ is then applied to the grid of feature vectors. The masks are such that the receptive field of each resulting *context vector* $c_{i,j}$ only includes feature vectors that lie above it in the image (i.e. $c_{i,j} = g_\phi(\{\boldsymbol{z}_{u,v}\}_{u \leq i,v})$). The prediction task then consists of predicting 'future' feature vectors $z_{i+k,j}$ from current context vectors $c_{i,j}$, where $k > 0$. The predictions are made linearly: given a context vector $c_{i,j}$, a prediction length $k > 0$, and a prediction matrix $W_k$, the predicted feature vector is $\hat{\boldsymbol{z}}_{i+k,j} = \boldsymbol{W}_k \boldsymbol{c}_{i,j}$.

The quality of this prediction is then evaluated using a contrastive loss. Specifically, the goal is to correctly recognize the target $z_{i+k,j}$ among a set of randomly sampled feature vectors $\{z_l\}$ from the dataset. We compute the probability assigned to the target using a softmax, and rate this probability using the usual cross-entropy loss. Summing this loss over locations and prediction offsets, we arrive at the CPC objective as defined in [135]:

$$\begin{aligned}\mathcal{L}_{\text{CPC}} &= -\sum_{i,j,k} \log p(\boldsymbol{z}_{i+k,j} | \hat{\boldsymbol{z}}_{i+k,j}, \{\boldsymbol{z}_l\}) \\ &= -\sum_{i,j,k} \log \frac{\exp(\hat{\boldsymbol{z}}_{i+k,j}^T \boldsymbol{z}_{i+k,j})}{\exp(\hat{\boldsymbol{z}}_{i+k,j}^T \boldsymbol{z}_{i+k,j}) + \sum_l \exp(\hat{\boldsymbol{z}}_{i+k,j}^T \boldsymbol{z}_l)}\end{aligned}$$

The *negative samples* $\{\boldsymbol{z}_l\}$ are taken from other locations in the image and other images in the mini-batch. This loss is called InfoNCE as it is inspired by Noise-Contrastive Estimation [65, 132] and has been shown to maximize the mutual information between $\boldsymbol{c}_{i,j}$ and $\boldsymbol{z}_{i+k,j}$ [135].

### Evaluation protocol

Having trained an encoder network $f_\theta$, a context network $g_\phi$, and a set of linear predictors $\{\boldsymbol{W}_k\}$ using the CPC objective, we use the encoder to form a representation $\boldsymbol{z} = f_\theta(\boldsymbol{x})$ of new observations

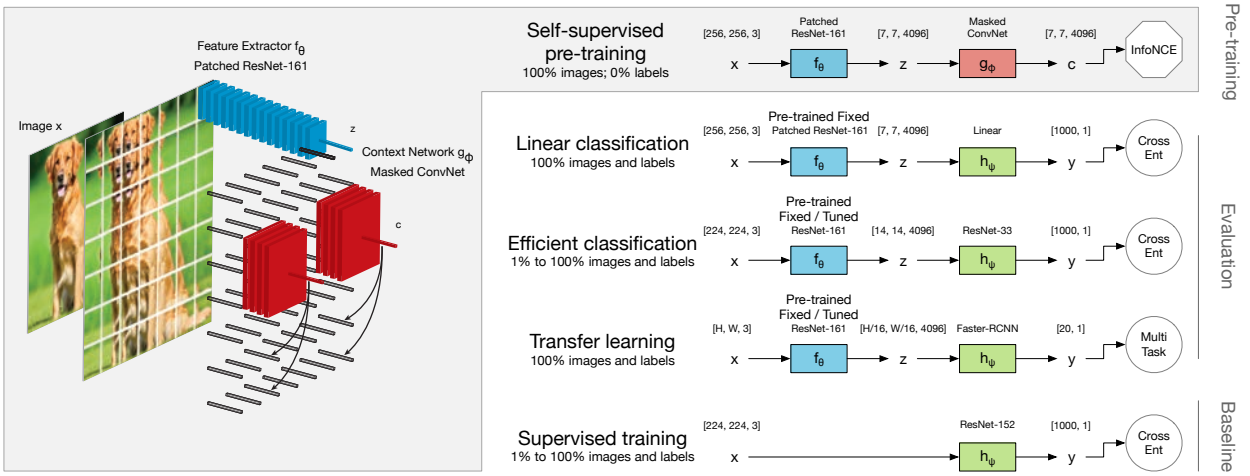



Figure 2.2: Overview of the framework for semi-supervised learning with Contrastive Predictive Coding. Left: unsupervised pre-training with the spatial prediction task (See Section 2.2). First, an image is divided into a grid of overlapping patches. Each patch is encoded independently from the rest with a feature extractor (blue) which terminates with a mean-pooling operation, yielding a single feature vector for that patch. Doing so for all patches yields a field of such feature vectors (wireframe vectors). Feature vectors above a certain level (in this case, the center of the image) are then aggregated with a context network (red), yielding a row of context vectors which are used to linearly predict features vectors below. Right: using the CPC representation for a classification task. Having trained the encoder network, the context network (red) is discarded and replaced by a classifier network (green) which can be trained in a supervised manner. In some experiments, we also fine-tune the encoder network (blue) for the classification task. When applying the encoder to cropped patches (as opposed to the full image) we refer to it as a *patched* ResNet in the figure.

$\boldsymbol{x}$, and discard the rest. Note that while pre-training required that the encoder be applied to patches, for downstream recognition tasks we can apply it directly to the entire image. We train a model $h_\psi$ to classify these representations: given a dataset of $N$ unlabeled images $\mathbb{D}_u = \{x_n\}$, and a (potentially much smaller) dataset of $M$ labeled images $\mathbb{D}_l = \{x_m, y_m\}$

$$\theta^* = \arg\min_\theta \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}_{\text{CPC}}[f_\theta(x_n)]$$

$$\psi^* = \arg\min_\psi \frac{1}{M} \sum_{m=1}^{M} \mathcal{L}_{\text{Sup}}[h_\psi \circ f_{\theta^*}(x_m), y_m]$$

In all cases, the dataset of unlabeled images $\mathbb{D}_u$ we pre-train on is the full ImageNet ILSVRC 2012 training set [150]. We consider three labeled datasets $\mathbb{D}_l$ for evaluation, each with an associated classifier $h_\psi$ and supervised loss $\mathcal{L}_{\text{Sup}}$ (see Fig. 2.2, right). This protocol is sufficiently generic to allow us to later compare the CPC representation to other methods which have their own means of learning a feature extractor $f_\theta$.

**Linear classification** is a standard benchmark for evaluating the quality of unsupervised image representations. In this regime, the classification network $h_\psi$ is restricted to mean pooling followed by a single linear layer, and the parameters of $f_\theta$ are kept fixed. The labeled dataset $\mathbb{D}_l$ is the entire ImageNet dataset, and the supervised loss $\mathcal{L}_{\text{Sup}}$ is standard cross-entropy. We use the same data-augmentation as in the unsupervised learning phase for training, and none at test time and evaluate with a single crop.

**Efficient classification** directly tests whether the CPC representation enables generalization from few labels. For this task, the classifier $h_\psi$ is an arbitrary deep neural network (we use an 11-block ResNet architecture [73] with 4096-dimensional feature maps and 1024-dimensional bottleneck layers). The labeled dataset $\mathbb{D}_l$ is a random subset of the ImageNet dataset: we investigated using $1\%$, $2\%$, $5\%$, $10\%$, $20\%$, $50\%$ and $100\%$ of the dataset. The supervised loss $\mathcal{L}_{\text{Sup}}$ is again cross-entropy. We use the same data-augmentation as during unsupervised pre-training, none at test-time and evaluate with a single crop.

**Transfer learning** tests the generality of the representation by applying it to a new task and dataset. For this we chose object detection on the PASCAL VOC 2007 dataset, a standard benchmark in computer vision [50]. As such $\mathbb{D}_l$ is the entire PASCAL VOC 2007 dataset (comprised of 5011 labeled images); $h_\psi$ and $\mathcal{L}_{\text{Sup}}$ are the Faster-RCNN architecture and loss [147]. In addition to color-dropping, we use the scale-augmentation from [39] for training.

For **linear classification**, we keep the feature extractor $f_\theta$ *fixed* to assess the representation in absolute terms. For **efficient classification** and **transfer learning**, we additionally explore *fine-tuning* the feature extractor for the supervised objective. In this regime, we initialize the feature extractor and classifier with the solutions $\theta^*, \psi^*$ found in the previous learning phase, and train them both for the supervised objective. To ensure that the feature extractor does not deviate too much from the solution dictated by the CPC objective, we use a smaller learning rate and early-stopping.

## 2.3 Related Work

Data-efficient learning has typically been approached by two complementary methods, both of which seek to make use of more plentiful unlabeled data: representation learning and label propagation. The former formulates an objective to learn a feature extractor $f_\theta$ in an unsupervised manner, whereas the latter directly constrains the classifier $h_\psi$ using the unlabeled data.

**Representation learning** saw early success using generative modeling [101], but likelihood-based models have yet to generalize to more complex stimuli. Generative adversarial models have also been harnessed for representation learning [41], and large-scale implementations have led to corresponding gains in linear classification accuracy [42].

In contrast to generative models which require the reconstruction of observations, self-supervised techniques directly formulate tasks involving the learned representation. For example, simply asking a network to recognize the spatial layout of an image led to representations that transferred to popular vision tasks such as classification and detection [39, 134]. Other works showed that prediction of color [207, 109] and image orientation [57], and invariance to data augmentation [44] can provide useful self-supervised tasks. Beyond single images, works have leveraged video cues such as object

tracking [190], frame ordering [130], and object boundary cues [117, 137]. Non-visual information can be equally powerful: information about camera motion [1, 93], scene geometry [204], or sound [5, 6] can all serve as natural sources of supervision.

While many of these tasks require predicting fixed quantities computed from the data, another class of *contrastive* methods [29, 68] formulate their objectives in the learned representations themselves. CPC is a contrastive representation learning method that maximizes the mutual information between spatially removed latent representations with InfoNCE [135], a loss function based on Noise-Contrastive Estimation [65, 132]. Two other methods have recently been proposed using the same loss function, but with different associated prediction tasks. Contrastive Multiview Coding [180] maximizes the mutual information between representations of different views of the same observation. Augmented Multiscale Deep InfoMax (AMDIM, [9]) is most similar to CPC in that it makes predictions across space, but differs in that it also predicts representations across layers in the model. Instance Discrimination is another contrastive objective which encourages representations that can discriminate between individual examples in the dataset [196].

A common alternative approach for improving data efficiency is **label-propagation** [209], where a classifier is trained on a subset of labeled data, then used to label parts of the unlabeled dataset. This label-propagation can either be discrete (as in pseudo-labeling, [115]) or continuous (as in entropy minimization, [63]). The predictions of this classifier are often constrained to be smooth with respect to certain deformations, such as data-augmentation [199] or adversarial perturbation [131]. Representation learning and label propagation have been shown to be complementary and can be combined to great effect [205], hence we focus solely on representation learning in this work.

## 2.4 Results

When testing whether CPC enables data-efficient learning, we wish to use the best representative of this model class. Unfortunately, purely unsupervised metrics tell us little about downstream performance, and implementation details have been shown to matter enormously [40, 102]. Since most representation learning methods have previously been evaluated using linear classification, we use this benchmark to guide a series of modifications to the training protocol and architecture (section 2.4) and compare to published results. In section 2.4 we turn to our central question of whether CPC enables data-efficient classification. Finally, in section 2.4 we investigate the generality of our results through transfer learning to PASCAL VOC 2007.

### From CPC v1 to CPC v2

The overarching principle behind our new model design is to increase the scale and efficiency of the encoder architecture while also maximizing the supervisory signal we obtain from each image. At the same time, it is important to control the types of predictions that can be made across image patches, by removing low-level cues which might lead to degenerate solutions. To this end, we augment individual patches independently using stochastic data-processing techniques from supervised and self-supervised learning.

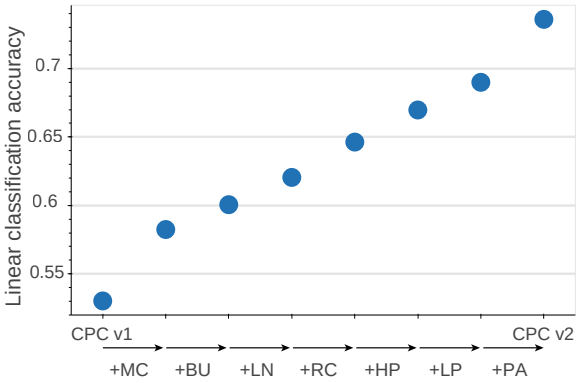



Figure 2.3: Linear classification performance of new variants of CPC, which incrementally add a series of modifications. MC: model capacity. BU: bottom-up spatial predictions. LN: layer normalization. RC: random color-dropping. HP: horizontal spatial predictions. LP: larger patches. PA: further patch-based augmentation. Note that these accuracies are evaluated on a custom validation set and are therefore not directly comparable to the results we report on the official validation set.

We identify four axes for model capacity and task setup that could impact the model's performance. The first axis increases model capacity by increasing depth and width, while the second improves training efficiency by introducing layer normalization. The third axis increases task complexity by making predictions in all four directions, and the fourth does so by performing more extensive patch-based augmentation.

**Model capacity.** Recent work has shown that larger networks and more effective training improves self-supervised learning [40, 102], but the original CPC model used only the first 3 stacks of a ResNet-101 architecture. Therefore, we convert the third residual stack of the ResNet-101 (containing 23 blocks, 1024-dimensional feature maps, and 256-dimensional bottleneck layers) to use 46 blocks with 4096-dimensional feature maps and 512-dimensional bottleneck layers. We call the resulting network ResNet-161. Consistent with prior results, this new architecture delivers better performance without any further modifications (Fig. 2.3, **+5%** Top-1 accuracy). We also increase the model's expressivity by increasing the size of its receptive field with larger patches (from 64×64 to 80×80 pixels; **+2%** Top-1 accuracy).

**Layer normalization.** Large architectures are more difficult to train efficiently. Early works on context prediction with patches used batch normalization [90, 39] to speed up training. However, with CPC we find that batch normalization actually harms downstream performance of large models. We hypothesize that batch normalization allows these models to find a trivial solution to CPC: it introduces a dependency between patches (through the batch statistics) that can be exploited to bypass the constraints on the receptive field. Nevertheless we find that we can reclaim much of batch normalization's training efficiency by using layer normalization (**+2%** accuracy, [8]).

Table 2.1: Linear classification accuracy, and comparison to other self-supervised methods. In all cases the feature extractor is optimized in an unsupervised manner, using one of the methods listed below. A linear classifier is then trained on top using all labels in the ImageNet dataset, and evaluated using a single crop. Prior art reported from [1] [196], [2] [210], [3] [76], [4] [129], [5] [40], [6] [102], [7] [135], [8] [42], [9] [9], [10] [180].

| Method | Params (M) | Top-1 | Top-5 |
|---|---|---|---|
| *Methods using ResNet-50:* | | | |
| Instance Discr. [1] | 24 | 54.0 | - |
| Local Aggr. [2] | 24 | 58.8 | - |
| MoCo [3] | 24 | 60.6 | - |
| PIRL [4] | 24 | 63.6 | - |
| CPC v2 - ResNet-50 | 24 | **63.8** | **85.3** |
| *Methods using different architectures:* | | | |
| Multi-task [5] | 28 | - | 69.3 |
| Rotation [6] | 86 | 55.4 | - |
| CPC v1 [7] | 28 | 48.7 | 73.6 |
| BigBiGAN [8] | 86 | 61.3 | 81.9 |
| AMDIM [9] | 626 | 68.1 | - |
| CMC [10] | 188 | 68.4 | 88.2 |
| MoCo [2] | 375 | 68.6 | - |
| CPC v2 - ResNet-161 | 305 | **71.5** | **90.1** |

**Prediction lengths and directions.** Larger architectures also run a greater risk of overfitting. We address this by asking more from the network: specifically, whereas the model in [135] predicted each patch using only context from above, we repeatedly predict the same patch using context from below, the right and the left (using separate context networks), resulting in up to four times as many prediction tasks. Additional predictions tasks incrementally increased accuracy (adding bottom-up predictions: **+2%** accuracy; using all four spatial directions: **+2.5%** accuracy).

**Patch-based augmentation.** If the network can solve CPC using low-level patterns (e.g. straight lines continuing between patches or chromatic aberration), it need not learn semantically meaningful content. Augmenting the low-level variability across patches can remove such cues. To that effect, the original CPC model spatially jittered individual patches independently. We further this logic by adopting the 'color dropping' method of [39], which randomly drops two of the three color channels in each patch, and find it to deliver systematic gains (**+3%** accuracy). We therefore continued by adding a fixed, generic augmentation scheme using the primitives from [33] (e.g. shearing,

Table 2.2: Data-efficient image classification. We compare the accuracy of two ResNet classifiers, one trained on the raw image pixels, the other on the proposed CPC v2 features, for varying amounts of labeled data. Note that we also fine-tune the CPC features for the supervised task, given the limited amount of labeled data. Regardless, the ResNet trained on CPC features systematically surpasses the one trained on pixels, even when given 2–5× less labels to learn from. The red (respectively, blue) boxes highlight comparisons between the two classifiers, trained with different amounts of data, which illustrate a 5× (resp. 2×) gain in data-efficiency in the low-data (resp. high-data) regime.

| Labeled data | 1% | 2% | 5% | 10% | 20% | 50% | 100% |
|---|---|---|---|---|---|---|---|
| | | | | Top-1 accuracy | | | |
| ResNet-200 trained on pixels | 23.1 | 34.8 | 50.6 | 62.5 | 70.3 | 75.9 | 80.2 |
| ResNet-33 trained on CPC features | 52.7 | 60.4 | 68.1 | 73.1 | 76.7 | 81.2 | 83.4 |
| Gain in data-efficiency | 5× | 2.5× | 2× | 2× | 2.5× | 2× | |
| | | | | Top-5 accuracy | | | |
| ResNet-200 trained on pixels | 44.1 | 59.9 | 75.2 | 83.9 | 89.4 | 93.1 | 95.2 |
| ResNet-33 trained on CPC features | 78.3 | 83.9 | 88.8 | 91.2 | 93.3 | 95.6 | 96.5 |
| Gain in data-efficiency | 5× | 5× | 2× | 2.5× | 2× | 2× | |

rotation, etc), as well as random elastic deformations and color transforms ([36], **+4.5%** accuracy in total). Note that these augmentations introduce some inductive bias about content-preserving transformations in images, but we do not optimize them for downstream performance (as in [33] and [119]).

**Comparison to previous art.** Cumulatively, these fairly straightforward implementation changes lead to a substantial improvement to the original CPC model, setting a new state-of-the-art in linear classification of 71.5% Top-1 accuracy (compared to 48.7% for the original, see table 2.1). Note that our architecture differs from ones used by other works in self-supervised learning, while using a number of parameters which is comparable to recently-used ones. The great diversity of network architectures (e.g. BigBiGAN employs a RevNet-50 with a $\times 4$ widening factor, AMDIM a customized ResNet architecture, CMC a ResNet-50 $\times 2$ and Momentum Contrast and ResNet-50 $\times 4$) make any apples-to-apples comparison with these works challenging. In order to compare with published results which use the same architecture, we therefore also trained a ResNet-50 architecture for the CPC v2 objective, arriving at 63.8% linear classification accuracy. This model outperforms methods which use the same architecture, as well as many recent approaches which at times use substantially larger ones [40, 135, 102, 210, 42].

## Efficient image classification

We now turn to our original question of whether CPC can enable data-efficient image recognition.

**Supervised baseline.** We start by evaluating the performance of purely-supervised networks as the size of the labeled dataset $\mathbb{D}_l$ varies from 1% to 100% of ImageNet, training separate classifiers on each subset. We compared a range of different architectures (ResNet-50, -101, -152, and -200) and found a ResNet-200 to work best across all data-regimes. After tuning the supervised model for low-data classification (varying network depth, regularization, and optimization parameters) and extensive use of data-augmentation (including the transformations used for CPC pre-training), the accuracy of the best model reaches 44.1% Top-5 accuracy when trained on 1% of the dataset (compared to 95.2% when trained on the entire dataset, see Table 2.2 and Fig. 4.2, red).

**Contrastive Predictive Coding.** We now address our central question of whether CPC enables data-efficient learning. We follow the same paradigm as for the supervised baseline (training and evaluating a separate classifier for each labeled subset), stacking a neural network classifier on top of the CPC latents $\boldsymbol{z} = f_\theta(\boldsymbol{x})$ rather than the raw image pixels $\boldsymbol{x}$. Specifically, we stack an 11-block ResNet classifier $h_\psi$ on top of the 14×14 grid of CPC latents, and train it using the same protocol as the supervised baseline (see section 2.2). During an initial phase we keep the CPC feature extractor fixed and train the ResNet classifier till convergence (see Table 2.3 for its performance). We then fine-tune the entire stack $h_\psi \circ f_\theta$ for the supervised objective, for a small number of epochs (chosen by cross-validation). In Table 2.2 and Fig. 4.2 (blue curve) we report the results of this fine-tuned model.

This procedure leads to a substantial increase in accuracy, yielding 78.3% Top-5 accuracy with only 1% of the labels, a 34% absolute improvement (77% relative) over purely-supervised methods. Surprisingly, when given the entire dataset, this classifier reaches 83.4%/96.5% Top1/Top5 accuracy, surpassing our supervised baseline (ResNet-200: 80.2%/95.2% accuracy) and published results (original ResNet-200 v2: 79.9%/95.2%, [74]; with AutoAugment: 80.0%/95.0%, [33]). Using this representation also leads to gains in data-efficiency. With only 50% of the labels our classifier surpasses the supervised baseline given the entire dataset, representing a 2× gain in data-efficiency (see table 2.2, blue boxes). Similarly, with only 1% of the labels, our classifier surpasses the supervised baseline given 5% of the labels (i.e. a 5× gain in data-efficiency, see table 2.2, red boxes).

Note that we are comparing two different model *classes* as opposed to specific models or instantiations of these classes. As result we have searched for the best representative of each class, landing on the ResNet-200 for purely supervised ResNets and our wider ResNet-161 for CPC pre-training (with a ResNet-33 for downstream classification). Given the difference in capacity between these models (the ResNet-200 has approximately 60 million parameters whereas our combined model has over 500 million parameters), we verified that supervised learning would not benefit from this larger architecture. Training the ResNet-161 + ResNet-33 stack (including batch normalization throughout) in a purely supervised manner yielded results that were similar to that of the ResNet-200 (80.3%/95.2% Top-1/Top-5 accuracy). This result is to be expected: the family of

Table 2.3: Comparison to other methods for semi-supervised learning. *Representation learning* methods use a classifier to discriminate an unsupervised representation, and optimize it solely with respect to labeled data. *Label-propagation* methods on the other hand further constrain the classifier with smoothness and entropy criteria on unlabeled data, making the additional assumption that all training images fit into a single (unknown) testing category. When evaluating CPC v2, BigBiGAN, and AMDIM, we train a ResNet-33 on top of the representation, while keeping the representation *fixed* or allowing it to be *fine-tuned*. All other results are reported from their respective papers: [1] [205], [2] [199], [3] [196], [4] [129].

| Labeled data | 1% | 10% | 100% |
|---|---|---|---|
| | Top-5 accuracy | | |
| Supervised baseline | 44.1 | 83.9 | 95.2 |
| *Methods using label-propagation:* | | | |
| Pseudolabeling [1] | 51.6 | 82.4 | - |
| VAT + Entropy Min. [1] | 47.0 | 83.4 | - |
| Unsup. Data Aug. [2] | - | 88.5 | - |
| Rot. + VAT + Ent. Min. [1] | - | **91.2** | 95.0 |
| *Methods using representation learning only:* | | | |
| Instance Discr. [3] | 39.2 | 77.4 | - |
| PIRL [4] | 57.2 | 83.8 | - |
| Rotation [1] | 57.5 | 86.4 | - |
| BigBiGAN (fixed) | 55.2 | 78.8 | 87.0 |
| AMDIM (fixed) | 67.4 | 85.8 | 92.2 |
| CPC v2 (fixed) | 77.1 | 90.5 | 96.2 |
| CPC v2 (fine-tuned) | **78.3** | **91.2** | **96.5** |

ResNet-50, -101, and -200 architectures are designed for supervised learning, and their capacity is calibrated for the amount of training signal present in ImageNet labels; larger architectures only run a greater risk of overfitting. In contrast, the CPC training objective is much richer and requires larger architectures to be taken advantage of, as evidenced by the difference in linear classification accuracy between a ResNet-50 and ResNet-161 trained for CPC (table 1, 63.8% vs 71.5% Top-1 accuracy).

**Other unsupervised representations.** How well does the CPC representation compare to other representations that have been learned in an unsupervised manner? Table 2.3 compares our best model with other works on efficient recognition. We consider three objectives from different model classes: self-supervised learning with rotation prediction [205], large-scale adversarial feature

learning (BigBiGAN, [42]), and another contrastive prediction objective (AMDIM, [9]). [205] evaluate the low-data classification performance of representations learned with rotation prediction using a similar paradigm and architecture (ResNet-152 with a $\times 2$ widening factor), hence we report their results directly: given 1% of ImageNet labels, their method achieves 57.5% Top-5 accuracy. The authors of BigBiGAN and AMDIM do not report results on efficient classification, hence we evaluated these representations using the same paradigm we used for evaluating CPC. Specifically, since fine-tuned representations yield only marginal gains over fixed ones (e.g. 77.1% vs 78.3% Top-5 accuracy given 1% of the labels, see table 2.3), we train an identical ResNet classifier on top of these representations while keeping them fixed. Given 1% of ImageNet labels, classifiers trained on top of BigBiGAN and AMDIM achieve 55.2% and 67.4% Top-5 accuracy, respectively.

Finally, Table 2.3 (top) also includes results for label-propagation algorithms. Note that the comparison is imperfect: these methods have an advantage in assuming that all unlabeled images can be assigned to a single category. At the same time, prior works (except for [205] which use a ResNet-50 $\times 4$) report results with smaller networks, which may degrade performance relative to ours. Overall, we find that our results are on par with or surpass even the strongest such results [205], even though this work combines a variety of techniques (entropy minimization, virtual adversarial training, self-supervised learning, and pseudo-labeling) with a large architecture whose capacity is similar to ours.

In summary, we find that CPC provides gains in data-efficiency that were previously unseen from representation learning methods, and rival the performance of the more elaborate label-propagation algorithms.

### Transfer learning: image detection on PASCAL VOC 2007

We next investigate transfer learning performance on object detection on the PASCAL VOC 2007 dataset, which reflects the practical scenario where a representation must be trained on a dataset with different statistics than the dataset of interest. This dataset also tests the efficiency of the representation as it only contains 5011 labeled images to train from. The standard protocol in this setting is to train an ImageNet classifier in a supervised manner, and use it as a feature extractor for a Faster-RCNN object detection architecture [147]. Following this procedure, we obtain 74.7% mAP with a ResNet-152 (Table 2.4). In contrast, if we use our CPC encoder as a feature extractor in the same setup, we obtain 76.6% mAP. This represents one of the first results where unsupervised pre-training surpasses supervised pre-training for transfer learning. Note that consistently with the previous section, we limit ourselves to comparing the two model *classes* (supervised vs. self-supervised), choosing the best architecture for each. Concurrently with our results, [76] achieve 74.9% in the same setting.

## 2.5 Discussion

We asked whether CPC could enable data-efficient image recognition, and found that it indeed greatly improves the accuracy of classifiers and object detectors when given small amounts of

Table 2.4: Comparison of PASCAL VOC 2007 object detection accuracy to other transfer methods. The supervised baseline learns from the entire labeled ImageNet dataset and fine-tunes for PASCAL detection. The second class of methods learns from the same *unlabeled* images before transferring. The architecture column specifies the object detector (Fast-RCNN or Faster-RCNN) and the feature extractor (ResNet-50, -101, -152, or -161). All of these methods pre-train on the ImageNet dataset, except for DeeperCluster which learns from the larger, but uncurated, YFCC100M dataset [179]. All methods fine-tune on the PASCAL 2007 training set, and are evaluted in terms of mean average precision (mAP). Prior art reported from [1] [44], [2] [40], [3] [137], [4] [207], [5] [39], [6] [196], [7] [24], [8] [25], [9] [210], [10] [129] [11] [76].

| Method | Architecture | mAP |
|---|---|---|
| *Transfer using labeled data:* | | |
| Supervised baseline | Faster: R152 | 74.7 |
| *Transfer using unlabeled data:* | | |
| Exemplar [1] by [2] | Faster: R101 | 60.9 |
| Motion Segm. [3] by [2] | Faster: R101 | 61.1 |
| Colorization [4] by [2] | Faster: R101 | 65.5 |
| Relative Pos. [5] by [2] | Faster: R101 | 66.8 |
| Multi-task [2] | Faster: R101 | 70.5 |
| Instance Discr. [6] | Faster: R50 | 65.4 |
| Deep Cluster [7] | Fast: VGG-16 | 65.9 |
| Deeper Cluster [8] | Fast: VGG-16 | 67.8 |
| Local Aggregation [9] | Faster: R50 | 69.1 |
| PIRL [10] | Faster: R50 | 73.4 |
| Momentum Contrast [11] | Faster: R50 | 74.9 |
| CPC v2 | Faster: R161 | **76.6** |

labeled data. Surprisingly, CPC even improves their peformance when given ImageNet-scale labels. Our results show that there is still room for improvement using relatively straightforward changes such as augmentation, optimization, and network architecture. Overall, these results open the door toward research on problems where data is naturally limited, e.g. medical imaging or robotics.

Furthermore, images are far from the only domain where unsupervised representation learning is important: for example, unsupervised learning is already a critical step in natural language processing [127, 38], and shows promise in domains like audio [135, 6, 5], video [94, 130], and robotic manipulation [140, 139, 158]. Currently much self-supervised work builds upon tasks tailored for a specific domain (often images), which may not be easily adapted to other domains. Contrastive prediction methods, including the techniques proposed in this paper, are task agnostic

and could therefore serve as a unifying framework for integrating these tasks and modalities. This generality is particularly useful given that many real-world environments are inherently multimodal, e.g. robotic environments which can have vision, audio, touch, proprioception, action, and more over long temporal sequences. Given the importance of increasing the amounts of self-supervision (via additional prediction tasks), integrating these modalities and tasks could lead to unsupervised representations which rival the efficiency and effectiveness of human ones.

## 2.6 Self-supervised pre-training

**Model architecture:** Having extracted 80×80 patches with a stride of 36×36 from an input image with 260×260 resolution, we end up with a grid of 6×6 image patches. We transform each one with a ResNet-161 encoder which terminates with a mean pooling operation, resulting in a [6,6,4096] tensor representation for each image. We then aggregate these `latents` into a 6×6 grid of context vectors, using a `pixelCNN`. We use this context to make the predictions and compute the `CPC` loss.

```
def pixelCNN(latents):
    # latents: [B, H, W, D]
    cres = latents
    cres_dim = cres.shape[-1]
    for _ in range(5):
      c = Conv2D(output_channels=256,
                 kernel_shape=(1, 1))(cres)
      c = ReLU(c)
      c = Conv2D(output_channels=256,
                 kernel_shape=(1, 3))(c)
      c = Pad(c, [[0, 0], [1, 0], [0, 0], [0, 0]])
      c = Conv2D(output_channels=256,
                 kernel_shape=(2, 1),
                 type='VALID')(c)
      c = ReLU(c)
      c = Conv2D(output_channels=cres_dim,
                 kernel_shape=(1, 1))(c)
      cres = cres + c
    cres = ReLU(cres)
    return cres

def CPC(latents, target_dim=64, emb_scale=0.1,
        steps_to_ignore=2, steps_to_predict=3):
    # latents: [B, H, W, D]
    loss = 0.0
    context = pixelCNN(latents)
    targets = Conv2D(output_channels=target_dim,
                     kernel_shape=(1, 1))(latents)
    batch_dim, col_dim, rows = targets.shape[:-1]
    targets = reshape(targets, [-1, target_dim])
    for i in range(steps_to_ignore, steps_to_predict):
      col_dim_i = col_dim - i - 1
      total_elements = batch_dim * col_dim_i * rows

      preds_i = Conv2D(output_channels=target_dim,
                       kernel_shape=(1, 1))(context)
      preds_i = preds_i[:, :-(i+1), :, :] * emb_scale
```

```
    preds_i = reshape(preds_i, [-1, target_dim])

    logits = matmul(preds_i, targets, transp_b=True)

    b = range(total_elements) / (col_dim_i * rows)
    col = range(total_elements) % (col_dim_i * rows)
    labels = b * col_dim * rows + (i+1) * rows + col

    loss += cross_entropy_with_logits(logits, labels)
  return loss
```

**Image preprocessing:** The final CPC v2 image processing pipeline we adopt consists of the following steps. We first resize the image to 300×300 pixels and randomly extract a 260×260 pixel crop, then divide this image into a 6×6 grid of 80×80 patches. Then, for every patch:

1. Randomly choose two transformations from [33] and apply them using default parameters.
2. Using the primitives from [36], randomly apply elastic deformation and shearing with a probability of 0.2. Randomly apply their color-histogram autementations with a probability of 0.2.
3. Randomly apply the color augmentations from [173] with a probability of 0.8.
4. Randomly project the image to grey-scale with a probability of 0.25.

**Optimization details:** We train the network for the CPC objective using the Adam optimizer [99] for 200 epochs, using a learning rate of 0.0004, $\beta_1 = 0.8$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$ and Polyak averaging with a decay of 0.9999. We also clip gradients to have a maximum norm of 0.01. We train the model with a batch size of 512, which we spread across 32 workers.

## 2.7 Linear classification

**Model architecture:** For linear classification we encode each image in the same way as during self-supervised pre-training (section 2.6), yielding a 6×6 grid of 4096-dimensional features vectors. We then use Batch-Normalization [90] to normalize the features (omitting the scale parameter) followed by a 1×1 convolution mapping each feature in the grid to the 1000 logits for ImageNet classification. We then spatially mean-pool these logits to end up with the final log probabilities for the linear classification.

**Image preprocessing:** We use the same data pipeline as for self-supervised pre-training (section 2.6).

**Optimization details:** We use the Adam optimizer with a learning rate of 0.0005. We train the model with a batch size of 512 images spread over 16 workers.

## 2.8 Efficient classification

### Purely supervised

**Model architecture:** We investigate using ResNet-50, ResNet-101, ResNet-152, and ResNet-200 model architectures, all of them using the 'v2' variant [74], and find larger architectures to perform better, even when given smaller amounts of data. We insert a DropOut layer before the final linear classification layer [167].

**Image preprocessing:** We extract a randomly sized crop, as in the augmentations of [173]. We follow this with the same image transformations as for self-supervised pre-training (steps 1–4).

**Optimization details:** We use stochastic gradient descent, varying the learning rate in {0.05, 0.1, 0.2}, the weight decay logarithmically from $10^{-5}$ to $10^{-2}$, the DropOut linearly from 0 to 1, and the batch size per worker in {16, 32}. We search for the best-performing model separately for each subset of labeled training data, as more labeled data requires less regularization. Having chosen these hyperparameters using a separate validation set (approximately 10k images which we remove from the training set), we evaluate each model on the test set (i.e. the publicly available ILSVRC-2012 validation set).

### Semi-supervised with CPC

**Model architecture:** We apply the CPC encoder directly to the image, resulting in a $14 \times 14$ grid of feature vectors. These features are used as inputs to an 11-block ResNet classifier with 4096-dimensional hiddens layers and 1024-dimensional bottleneck layers. As for the supervised baseline, we insert DropOut after the final mean-pooling operation and before the final linear classifier.

**Image preprocessing:** We use the same pipeline as the supervised baseline.

**Optimization details:** We start by training the classifier while keeping the CPC features fixed. To do so we search through the same set of hyperparameters as the supervised baseline. After training the classifier till convergence, we fine-tune the entire stack for classification. In this phase we keep the optimization details of each component the same as previously: the classifier is fine-tuned with SGD, while the encoder is fine-tuned with Adam.

# Chapter 3

# Contrastive Learning for Reinforcement Learning

## 3.1 Introduction

Developing agents that can perform complex control tasks from high dimensional observations such as pixels has been possible by combining the expressive power of deep neural networks with the long-term credit assignment power of reinforcement learning algorithms. Notable successes include learning to play a diverse set of video games from raw pixels [133], continuous control tasks such as controlling a simulated car from a dashboard camera [118] and subsequent algorithmic developments and applications to agents that successfully navigate mazes and solve complex tasks from first-person camera observations [92, 48, 91]; and robots that successfully grasp objects in the real world [96].

However, it has been empirically observed that reinforcement learning from high dimensional observations such as raw pixels is sample-inefficient [107, 95]. Moreover, it is widely accepted that learning policies from physical state based features is significantly more sample-efficient than learning from pixels [177]. In principle, if the state information is present in the pixel data, then we should be able to learn representations that extract the relevant state information. For this reason, it may be possible to learn from pixels as fast as from state given the right representation.

From a practical standpoint, although high rendering speeds in simulated environments enable RL agents to solve complex tasks within reasonable wall clock time, learning in the real world means that agents are bound to work within the limitations of physics. [96] needed a farm of robotic arms that collected large scale robot interaction data over several months to develop their robot grasp value functions and policies. The data-efficiency of the whole pipeline thus has significant room for improvement. Similarly, in simulated worlds which are limited by rendering speeds in the absence of GPU accelerators, data efficiency is extremely crucial to have a fast experimental turnover and iteration. Therefore, improving the sample efficiency of reinforcement learning (RL) methods that operate from high dimensional observations is of paramount importance to RL research both in simulation and the real world and allows for faster progress towards the broader goal of developing

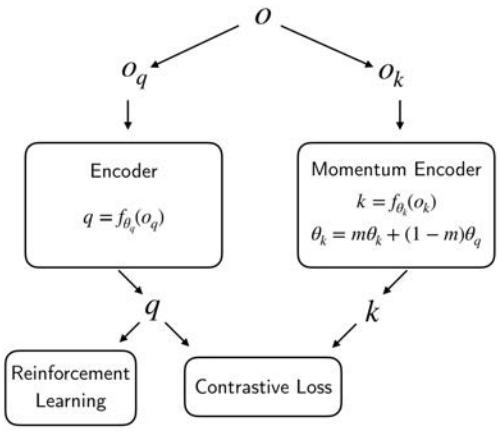



Figure 3.1: **C**ontrastive **U**nsupervised Representations for **R**einforcement **L**earning (CURL) combines instance contrastive learning and reinforcement learning. CURL trains a visual representation encoder by ensuring that the embeddings of data-augmented versions $o_q$ and $o_k$ of observation $o$ match using a contrastive loss. The *query* observations $o_q$ are treated as the anchor while the *key* observations $o_k$ contain the positive and negatives, all constructed from the minibatch sampled for the RL update. The keys are encoded with a momentum averaged version of the query encoder. The RL policy and (or) value function are built on top of the query encoder which is jointly trained with the contrastive and reinforcement learning objectives. CURL is a generic framework that can be plugged into any RL algorithm that relies on learning representations from high dimensional images.

intelligent autonomous agents.

A number of approaches have been proposed in the literature to address the sample inefficiency of deep RL algorithms. Broadly, they can be classified into two streams of research, though not mutually exclusive: (i) Auxiliary tasks on the agent's sensory observations; (ii) World models that predict the future. While the former class of methods use auxiliary self-supervision tasks to accelerate the learning progress of model-free RL methods [92, 128], the latter class of methods build explicit predictive models of the world and use those models to plan through or collect fictitious rollouts for model-free methods to learn from [171, 66, 95, 154].

Our work falls into the first class of models, which use auxiliary tasks to improve sample efficiency. Our hypothesis is simple: If an agent learns a useful semantic representation from high dimensional observations, control algorithms built on top of those representations should be significantly more data-efficient. Self-supervised representation learning has seen dramatic progress in the last couple of years with huge advances in masked language modeling [38] and contrastive learning [79, 76, 26] for language and vision respectively. The representations uncovered by these objectives improve the performance of any supervised learning system especially in scenarios where the amount of labeled data available for the downstream task is really low.

We take inspiration from the contrastive pre-training successes in computer vision. However,

there are a couple of key differences: (i) There is no giant unlabeled dataset of millions of images available beforehand - the dataset is collected online from the agent's interactions and changes dynamically with the agent's experience; (ii) The agent has to perform unsupervised and reinforcement learning simultaneously as opposed to fine-tuning a pre-trained network for a specific downstream task. These two differences introduce a different challenge: How can we use contrastive learning for improving agents that can learn to control effectively and efficiently from online interactions?

To address this challenge, we propose CURL - **C**ontrastive **U**unsupervised Representations for **R**einforcement **L**earning. CURL uses a form of contrastive learning that maximizes agreement between augmented versions of the same observation, where each observation is a stack of temporally sequential frames. We show that CURL significantly improves sample-efficiency over prior pixel-based methods by performing contrastive learning simultaneously with an off-policy RL algorithm. CURL coupled with the Soft-Actor-Critic (SAC) [67] results in **1.9x** median higher performance over Dreamer, a prior state-of-the-art algorithm on DMControl environments, benchmarked at **100k** environment steps and matches the performance of state-based SAC on the majority of 16 environments tested, a **first** for pixel-based methods. In the Atari setting benchmarked at 100k interaction steps, we show that CURL coupled with a data-efficient version of Rainbow DQN [72] results in **1.2x** median higher performance over prior methods such as SimPLe [95], improving upon Efficient Rainbow [72] on 19 out of 26 Atari games, surpassing human efficiency on two games.

While contrastive learning in aid of model-free RL has been studied in the past by [135] using Contrastive Predictive Coding (CPC), the results were mixed with marginal gains in a few DMLab [48] environments. CURL is the first model to show substantial data-efficiency gains from using a contrastive self-supervised learning objective for model-free RL agents across a multitude of pixel based continuous and discrete control tasks in DMControl and Atari.

We prioritize designing a simple and easily reproducible pipeline. While the promise of auxiliary tasks and learning world models for RL agents has been demonstrated in prior work, there's an added layer of complexity when introducing components like modeling the future in a latent space [135, 66]. CURL is designed to add minimal overhead in terms of architecture and model learning. The contrastive learning objective in CURL operates with the same latent space and architecture typically used for model-free RL and seamlessly integrates with the training pipeline without the need to introduce multiple additional hyperparameters.

Our paper makes the following **key contributions**: We present CURL, a simple framework that integrates contrastive learning with model-free RL with minimal changes to the architecture and training pipeline. Using 16 complex control tasks from the DeepMind control (DMControl) suite and 26 Atari games, we empirically show that contrastive learning combined with model-free RL outperforms the prior state-of-the-art by 1.9x on DMControl and 1.2x on Atari compared across leading prior pixel-based methods. CURL is also the first algorithm across both model-based and model-free methods that operates purely from pixels, and nearly matches the performance and sample-efficiency of a SAC algorithm trained from the state based features on the DMControl suite. Finally, our design is simple and does not require any custom architectural choices or hyperparameters which is crucial for reproducible end-to-end training. Through these strong empirical results, we demonstrate that a contrastive objective is the preferred self-supervised

auxiliary task for achieving sample-efficiency compared to reconstruction based methods, and enables model-free methods to outperform state-of-the-art model-based methods in terms of data-efficiency.

## 3.2 Related Work

**Self-Supervised Learning:** Self-Supervised Learning is aimed at learning rich representations of high dimensional unlabeled data to be useful for a wide variety of tasks. The fields of natural language processing and computer vision have seen dramatic advances in self-supervised methods such as BERT [38], CPC, MoCo, SimCLR [79, 76, 26].

**Contrastive Learning:** Contrastive Learning is a framework to learn representations that obey similarity constraints in a dataset typically organized by similar and dissimilar pairs. This is often best understood as performing a dictionary lookup task wherein the positive and negatives represent a set of keys with respect to a query (or an anchor). A simple instantiation of contrastive learning is Instance Discrimination [196] wherein a query and key are positive pairs if they are data-augmentations of the same instance (example, image) and negative otherwise. A key challenge in contrastive learning is the choice of negatives which can decide the quality of the underlying representations learned. The loss functions used to contrast could be among several choices such as InfoNCE [135], Triplet [190], Siamese [29] and so forth.

**Self-Supervised Learning for RL**: Auxiliary tasks such as predicting the future conditioned on the past observation(s) and action(s) [92, 160, 135, 153] are a few representative examples of using auxiliary tasks to improve the sample-efficiency of model-free RL algorithms. The future prediction is either done in a pixel space [92] or latent space [135]. The sample-efficiency gains from reconstruction-based auxiliary losses have been benchmarked in [92, 82, 202]. Contrastive learning has been used to extract reward signals in the latent space [158, 47, 193]; and study representation learning on Atari games by [3].

**World Models for sample-efficiency:** While joint learning of an auxiliary unsupervised task with model-free RL is one way to improve the sample-efficiency of agents, there has also been another line of research that has tried to learn world models of the environment and use them to sample rollouts and plan. An early instantiation of the generic principle was put forth by [171] in Dyna where fictitious samples rolled out from a learned world model are used in addition to the agent's experience for sample-efficient learning. Planning through a learned world model [166] is another way to improve sample-efficiency. While [92, 135, 114] also learn pixel and latent space forward models, the models are learned to shape the latent representations, and there is no explicit Dyna or planning. Planning through learned world models has been successfully demonstrated in [66, 70, 69]. [95] introduce SimPLe which implements Dyna with expressive deep neural networks for the world model for sample-efficiency on Atari games.

**Sample-efficient RL for image-based control:** CURL encompasses the areas of self-supervision, contrastive learning and using auxiliary tasks for sample-efficient RL. We benchmark for sample-efficiency on the DMControl suite [177] and Atari Games benchmarks [12]. The DMControl suite has been used widely by [202], [70], [69] and [114] for benchmarking sample-efficiency for image

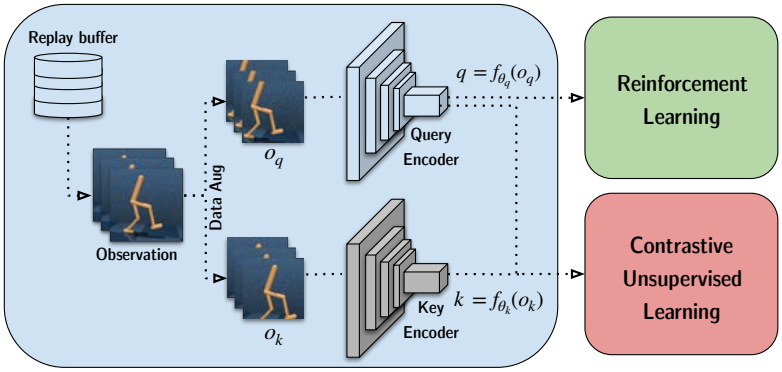



Figure 3.2: CURL Architecture: A batch of transitions is sampled from the replay buffer. Observations are then data-augmented twice to form query and key observations, which are then encoded with the query encoder and key encoders, respectively. The queries are passed to the RL algorithm while query-key pairs are passed to the contrastive learning objective. During the gradient update step, only the query encoder is updated. The key encoder weights are the moving average (EMA) of the query weights similar to MoCo [76].

based continuous control methods. As for Atari, [95] propose to use the 100k interaction steps benchmark for sample-efficiency which has been adopted in [98, 72]. The Rainbow DQN [81] was originally proposed for maximum sample-efficiency on the Atari benchmark and in recent times has been adapted to a version known as Data-Efficient Rainbow [72] with competitive performance to SimPLe without learning world models. We benchmark extensively against both model-based and model-free algorithms in our experiments. For the DMControl experiments, we compare our method to Dreamer, PlaNet, SLAC, SAC+AE whereas for Atari experiments we compare to SimPLe, Rainbow, and OverTrained Rainbow (OTRainbow) and Efficient Rainbow (Eff. Rainbow).

## 3.3 Background

CURL is a general framework for combining contrastive learning with RL. In principle, one could use any RL algorithm in the CURL pipeline, be it on-policy or off-policy. We use the widely adopted Soft Actor Critic (SAC) [67] for continuous control benchmarks (DM Control) and Rainbow DQN [81, 72] for discrete control benchmarks (Atari). Below, we review SAC, Rainbow DQN and Contrastive Learning.

### Soft Actor Critic

SAC is an off-policy RL algorithm that optimizes a stochastic policy for maximizing the expected trajectory returns. Like other state-of-the-art end-to-end RL algorithms, SAC is effective when solving tasks from state observations but fails to learn efficient policies from pixels. SAC is an actor-critic method that learns a policy $\pi_\psi$ and critics $Q_{\phi_1}$ and $Q_{\phi_2}$. The parameters $\phi_i$ are learned by minimizing the Bellman error:

$$\mathcal{L}(\phi_i, \mathcal{B}) = \mathbb{E}_{t\sim\mathcal{B}}\left[\left(Q_{\phi_i}(o,a) - (r + \gamma(1-d)\mathcal{T})\right)^2\right] \tag{3.1}$$

where $t = (o, a, o', r, d)$ is a tuple with observation $o$, action $a$, reward $r$ and done signal $d$, $\mathcal{B}$ is the replay buffer, and $\mathcal{T}$ is the target, defined as:

$$\mathcal{T} = \left(\min_{i=1,2} Q^*_{\phi_i}(o', a') - \alpha \log \pi_\psi(a'|o')\right) \tag{3.2}$$

In the target equation equation 3.2, $Q^*_{\phi_i}$ denotes the exponential moving average (EMA) of the parameters of $Q_{\phi_i}$. Using the EMA has empirically shown to improve training stability in off-policy RL algorithms. The parameter $\alpha$ is a positive entropy coefficient that determines the priority of the entropy maximization over value function optimization.

While the critic is given by $Q_{\phi_i}$, the actor samples actions from policy $\pi_\psi$ and is trained by maximizing the expected return of its actions as in:

$$\mathcal{L}(\psi) = \mathbb{E}_{a\sim\pi}\left[Q^\pi(o,a) - \alpha \log \pi_\psi(a|o)\right] \tag{3.3}$$

where actions are sampled stochastically from the policy $a_\psi(o, \xi) \sim \tanh\left(\mu_\psi(o) + \sigma_\psi(o) \odot \xi\right)$ and $\xi \sim \mathcal{N}(0, I)$ is a standard normalized noise vector.

## Rainbow

Rainbow DQN [81] is best summarized as multiple improvements on top of the original Nature DQN [133] applied together. Specifically, Deep Q Network (DQN) [133] combines the off-policy algorithm Q-Learning with a convolutional neural network as the function approximator to map raw pixels to action value functions. Since then, multiple improvements have been proposed such as Double Q Learning [185], Dueling Network Architectures [192], Prioritized Experience Replay [152], and Noisy Networks [51]. Additionally, distributional reinforcement learning [11] proposed the technique of predicting a distribution over possible value function bins through the C51 Algorithm. Rainbow DQN combines all of the above techniques into a single off-policy algorithm for state-of-the-art sample efficiency on Atari benchmarks. Additionally, Rainbow also makes use of multi-step returns [172]. [72] propose a data-efficient version of the Rainbow which can be summarized as an improved configuration of hyperparameters that is optimized for performance benchmarked at 100K interaction steps.

## Contrastive Learning

A key component of CURL is the ability to learn rich representations of high dimensional data using contrastive unsupervised learning. Contrastive learning [68, 112, 135, 196, 76] can be understood as learning a differentiable dictionary look-up task. Given a query $q$ and keys $\mathbb{K} = \{k_0, k_1, \dots\}$ and an explicitly known partition of $\mathbb{K}$ (with respect to $q$) $P(\mathbb{K}) = (\{k_+\}, \mathbb{K} \setminus \{k_+\})$, the goal of contrastive learning is to ensure that $q$ matches with $k_+$ relatively more than any of the keys in $\mathbb{K} \setminus \{k_+\}$. $q, \mathbb{K}, k_+$, and $\mathbb{K} \setminus \{k_+\}$ are also referred to as anchor, targets, positive, negatives respectively in the parlance of contrastive learning [135, 76]. Similarities between the anchor and

targets are best modeled with dot products ($q^T k$) [196, 76] or bilinear products ($q^T W k$) [135, 79] though other forms like euclidean distances are also common [155, 190]. To learn embeddings that respect these similarity relations, [135] propose the InfoNCE loss:

$$\mathcal{L}_q = \log \frac{\exp(q^T W k_+)}{\exp(q^T W k_+) + \sum_{i=0}^{K-1} \exp(q^T W k_i)} \tag{3.4}$$

The loss 3.4 can be interpreted as the log-loss of a $K$-way softmax classifier whose label is $k_+$.

## 3.4 CURL Implementation

CURL minimally modifies a base RL algorithm by training the contrastive objective as an auxiliary loss during the batch update. In our experiments, we train CURL alongside two model-free RL algorithms — SAC for DMControl experiments and Rainbow DQN (data-efficient version) for Atari experiments. To specify a contrastive learning objective, we need to define (i) the discrimination objective (ii) the transformation for generating query-key observations (iii) the embedding procedure for transforming observations into queries and keys and (iv) the inner product used as a similarity measure between the query-key pairs in the contrastive loss. The exact specification these aspects largely determine the quality of the learned representations.

We first summarize the CURL architecture, and then cover each architectural choice in detail.

### Architectural Overview

CURL uses instance discrimination with similarities to SimCLR [26], MoCo [76] and CPC [79]. Most Deep RL architectures operate with a stack of temporally consecutive frames as input [81]. Therefore, instance discrimination is performed across the frame stacks as opposed to single image instances. We use a momentum encoding procedure for targets similar to MoCo [77] which we found to be better performing for RL. Finally, for the InfoNCE score function, we use a bi-linear inner product similar to CPC [135] which we found to work better than unit norm vector products used in MoCo and SimCLR. Ablations for both the encoder and the similarity measure choices are shown in Figure 3.5. The contrastive representation is trained jointly with the RL algorithm, and the latent code receives gradients from both the contrastive objective and the Q-function. An overview of the architecture is shown in in Figure 3.2.

### Discrimination Objective

A key component of contrastive representation learning is the choice of positives and negative samples relative to an anchor [9, 180, 79, 76, 26]. Contrastive Predictive Coding (CPC) based pipelines [79, 135] use groups of image patches separated by a carefully chosen spatial offset for anchors and positives while the negatives come from other patches within the image and from other images.

While patches are a powerful way to incorporate spatial and instance discrimination together, they introduce extra hyperparameters and architectural design choices which may be hard to adapt for a new problem. SimCLR [26] and MoCo [76] opt for a simpler design where there is no patch extraction.

Discriminating transformed image instances as opposed to image-patches within the same image optimizes a simpler instance discrimination objective [196] with the InfoNCE loss and requires minimal architectural adjustments [77, 26]. It is preferable to pick a simpler discrimination objective in the RL setting for two reasons. First, considering the brittleness of reinforcement learning algorithms [80], complex discrimination may destabilize the RL objective. Second, since RL algorithms are trained on dynamically generated datasets, a complex discrimination objective may significantly increase the wall-clock training time. CURL therefore uses instance discrimination rather than patch discrimination. One could view contrastive instance discrimination setups like SimCLR and MoCo as maximizing mutual information between an image and its augmented version. The reader is encouraged to refer to [135, 85, 184] for connections between contrastive learning and mutual information.

## Query-Key Pair Generation

Similar to instance discrimination in the image setting [77, 26], the anchor and positive observations are two different augmentations of the same image while negatives come from other images. CURL primarily relies on the random crop data augmentation, where a random square patch is cropped from the original rendering.

A significant difference between RL and computer vision settings is that an instance ingested by a model-free RL algorithm that operates from pixels is not just a single image but a stack of frames [133]. For example, one typically feeds in a stack of 4 frames in Atari experiments and a stack of 3 frames in DMControl. This way, performing instance discrimination on frame stacks allows CURL to learn both spatial and temporal discriminative features. For details regarding the extent to which CURL captures temporal features, see Appendix 3.12.

We apply the random augmentations across the batch but consistently across each stack of frames to retain information about the temporal structure of the observation. The augmentation procedure is shown in Figure 3.3. For more details, refer to Appendix 3.8.

## Similarity Measure

Another determining factor in the discrimination objective is the inner product used to measure agreement between query-key pairs. CURL employs the bi-linear inner-product $\text{sim}(q, k) = q^T W k$, where $W$ is a learned parameter matrix. We found this similarity measure to outperform the normalized dot-product (see Figure 3.5 in Appendix 3.8) used in recent state-of-the-art contrastive learning methods in computer vision like MoCo and SimCLR.

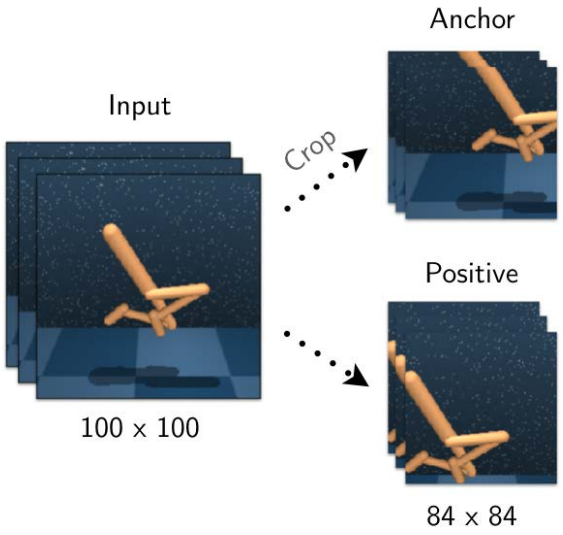



Figure 3.3: Visually illustrating the process of generating an anchor and its positive using stochastic random crops. Our aspect ratio for cropping is 0.84, i.e, we crop a $84 \times 84$ image from a $100 \times 100$ simulation-rendered image. Applying the same random crop coordinates across all frames in the stack ensures time-consistent spatial jittering.

## Target Encoding with Momentum

The motivation for using contrastive learning in CURL is to train encoders that map from high dimensional pixels to more semantic latents. InfoNCE is an unsupervised loss that learns encoders $f_q$ and $f_k$ mapping the raw anchors (query) $x_q$ and targets (keys) $x_k$ into latents $q = f_q(x_q)$ and $k = f_k(x_k)$, on which we apply the similarity dot products. It is common to share the same encoder between the anchor and target mappings, that is, to have $f_q = f_k$ [135, 79].

From the perspective of viewing contrastive learning as building differentiable dictionary lookups over high dimensional entities, increasing the size of the dictionary and enriching the set of negatives is helpful in learning rich representations. [76] propose momentum contrast (MoCo), which uses the exponentially moving average (momentum averaged) version of the query encoder $f_q$ for encoding the keys in $\mathbb{K}$. Given $f_q$ parametrized by $\theta_q$ and $f_k$ parametrized by $\theta_k$, MoCo performs the update $\theta_k = m\theta_k + (1-m)\theta_q$ and encodes any target $x_k$ using $\text{SG}(f_k(x_k))$ [SG : Stop Gradient].

CURL couples frame-stack instance discrimination with momentum encoding for the targets during contrastive learning, and RL is performed on top of the encoder features.

## Differences Between CURL and Prior Contrastive Methods in RL

[135] use Contastive Predictive Coding (CPC) as an auxiliary task wherein an LSTM operates on a latent space of a convolutional encoder; and both the CPC and A2C [133] objectives are jointly

optimized. CURL avoids using pipelines that predict the future in a latent space such as [135, 69]. In CURL, we opt for a simple instance discrimination style contrastive auxiliary task.

### CURL Contrastive Learning Pseudocode (PyTorch-like)

```
# f_q, f_k: encoder networks for anchor
# (query) and target (keys) respectively.
# loader: minibatch sampler from ReplayBuffer
# B-batch_size, C-channels, H,W-spatial_dims
# x : shape : [B, C, H, W]
# C = c * num_frames; c=3 (R/G/B) or 1 (gray)
# m: momentum, e.g. 0.95
# z_dim: latent dimension
f_k.params = f_q.params
W = rand(z_dim, z_dim) # bilinear product.
for x in loader: # load minibatch from buffer
 x_q = aug(x) # random augmentation
 x_k = aug(x) # different random augmentation
 z_q = f_q.forward(x_q)
 z_k = f_k.forward(x_k)
 z_k = z_k.detach() # stop gradient
 proj_k = matmul(W, z_k.T) # bilinear product
 logits = matmul(z_q, proj_k) # B x B
  # subtract max from logits for stability
 logits = logits - max(logits, axis=1)
 labels = arange(logits.shape[0])
 loss = CrossEntropyLoss(logits, labels)
 loss.backward()
 update(f_q.params) # Adam
 update(W) # Adam
 f_k.params = m*f_k.params+(1-m)*f_q.params
```

## 3.5 Experiments

### Evaluation

We measure the data-efficiency and performance of our method and baselines at 100k and 500k *environment steps* on DMControl and 100k *interaction steps* (400k environment steps with action repeat of 4) on Atari, which we will henceforth refer to as **DMControl100k**, **DMControl500k** and **Atari100k** for clarity. While Atari100k benchmark has been common practice when investigating data-efficiency on Atari [95, 72, 98], the DMControl benchmark was set at 500k environment steps because state-based RL approaches asymptotic performance on many environments at this point, and 100k steps to measure the speed of initial learning. A broader motivation is that while RL algorithms can achieve super-human performance on Atari games, they are still far less efficient than a human learner. Training for 100-500k environment steps corresponds to a few hours of human time.

We evaluate (i) *sample-efficiency* by measuring how many steps it takes the best performing baselines to match CURL performance at a fixed $T$ (100k or 500k) steps and (ii) *performance* by

measuring the ratio of the episode returns achieved by CURL versus the best performing baseline at $T$ steps. To be explicit, when we say data or sample-efficiency we're referring to (i) and when we say performance we're referring to (ii).

Table 3.1: Scores achieved by CURL (mean & standard deviation for 10 seeds) and baselines on DMControl500k and 1DMControl100k. CURL achieves state-of-the-art performance on the majority (**5** out of **6**) environments benchmarked on DMControl500k. These environments were selected based on availability of data from baseline methods (we run CURL experiments on 16 environments in total and show results in Figure 3.7). The baselines are PlaNet [70], Dreamer [69], SAC+AE [202], SLAC [114], pixel-based SAC and state-based SAC [67]. SLAC results were reported with one and three gradient updates per agent step, which we refer to as SLACv1 and SLACv2 respectively. We compare to SLACv1 since all other baselines and CURL only make one gradient update per agent step. We also ran CURL with three gradient updates per step and compare results to SLACv2 in Table 3.5.

| 500K step scores | CURL | PlaNet | Dreamer | SAC+AE | SLACv1 | Pixel SAC | State SAC |
|---|---|---|---|---|---|---|---|
| Finger, spin | **926 ± 45** | 561 ± 284 | 796 ± 183 | 884 ± 128 | 673 ± 92 | 179 ± 166 | 923 ± 21 |
| Cartpole, swingup | **841 ± 45** | 475 ± 71 | 762 ± 27 | 735 ± 63 | - | 419 ± 40 | 848 ± 15 |
| Reacher, easy | **929 ± 44** | 210 ± 390 | 793 ± 164 | 627 ± 58 | - | 145 ± 30 | 923 ± 24 |
| Cheetah, run | 518 ± 28 | 305 ± 131 | 570 ± 253 | 550 ± 34 | **640 ± 19** | 197 ± 15 | 795 ± 30 |
| Walker, walk | **902 ± 43** | 351 ± 58 | 897 ± 49 | 847 ± 48 | 842 ± 51 | 42 ± 12 | 948 ± 54 |
| Ball in cup, catch | **959 ± 27** | 460 ± 380 | 879 ± 87 | 794± 58 | 852 ± 71 | 312± 63 | 974 ± 33 |
| 100K step scores | | | | | | | |
| Finger, spin | **767 ± 56** | 136 ± 216 | 341 ± 70 | 740 ± 64 | 693 ± 141 | 179 ± 66 | 811±46 |
| Cartpole, swingup | **582±146** | 297±39 | 326±27 | 311±11 | - | 419±40 | 835±22 |
| Reacher, easy | **538±233** | 20±50 | 314±155 | 274±14 | - | 145±30 | 746±25 |
| Cheetah, run | 299 ±48 | 138±88 | 235± 137 | 267±24 | **319±56** | 197±15 | 616±18 |
| Walker, walk | **403±24** | 224±48 | 277±12 | 394±22 | 361±73 | 42±12 | 891±82 |
| Ball in cup, catch | **769 ± 43** | 0 ± 0 | 246 ± 174 | 391± 82 | 512 ± 110 | 312± 63 | 746±91 |

## Environments

Our primary goal for CURL is sample-efficient control from pixels that is broadly applicable across a range of environments. We benchmark the performance of CURL for both discrete and continuous control environments. Specifically, we focus on DMControl suite for continuous control tasks and the Atari Games benchmark for discrete control tasks with inputs being raw pixels rendered by the environments.

**DeepMind Control:** Recently, there have been a number of papers that have benchmarked for sample efficiency on challenging visual continuous control tasks belonging to the DMControl suite

Table 3.2: Scores achieved by CURL (coupled with Eff. Rainbow) and baselines on Atari benchmarked at 100k time-steps (Atari100k). CURL achieves state-of-the-art performance on **7** out of **26** environments. Our baselines are SimPLe [95], OverTrained Rainbow (OTRainbow) [98], Data-Efficient Rainbow (Eff. Rainbow) [72], Rainbow [81], Random Agent and Human Performance (Human). We see that CURL implemented on top of Eff. Rainbow improves over Eff. Rainbow on 19 out of 26 games. We also run CURL with 20 random seeds given that this benchmark is susceptible to high variance across multiple runs. We also see that CURL achieves superhuman performance on JamesBond and Krull.

| Game | Human | Random | Rainbow | SimPLe | OTRainbow | Eff. Rainbow | CURL |
|---|---|---|---|---|---|---|---|
| Alien | 7127.7 | 227.8 | 318.7 | 616.9 | **824.7** | 739.9 | 558.2 |
| Amidar | 1719.5 | 5.8 | 32.5 | 88.0 | 82.8 | **188.6** | 142.1 |
| Assault | 742.0 | 222.4 | 231 | 527.2 | 351.9 | 431.2 | **600.6** |
| Asterix | 8503.3 | 210.0 | 243.6 | **1128.3** | 628.5 | 470.8 | 734.5 |
| Bank Heist | 753.1 | 14.2 | 15.55 | 34.2 | **182.1** | 51.0 | 131.6 |
| Battle Zone | 37187.5 | 2360.0 | 2360.0 | 5184.4 | 4060.6 | 10124.6 | **14870.0** |
| Boxing | 12.1 | 0.1 | -24.8 | **9.1** | 2.5 | 0.2 | 1.2 |
| Breakout | 30.5 | 1.7 | 1.2 | **16.4** | 9.84 | 1.9 | 4.9 |
| Chopper Command | 7387.8 | 811.0 | 120.0 | **1246.9** | 1033.33 | 861.8 | 1058.5 |
| crazy_climber | 35829.4 | 10780.5 | 2254.5 | **62583.6** | 21327.8 | 16185.3 | 12146.5 |
| demon_attack | 1971.0 | 152.1 | 163.6 | 208.1 | 711.8 | 508.0 | **817.6** |
| freeway | 29.6 | 0.0 | 0.0 | 20.3 | 25.0 | **27.9** | 26.7 |
| frostbite | 4334.7 | 65.2 | 60.2 | 254.7 | 231.6 | 866.8 | **1181.3** |
| gopher | 2412.5 | 257.6 | 431.2 | 771.0 | **778.0** | 349.5 | 669.3 |
| hero | 30826.4 | 1027.0 | 487 | 2656.6 | 6458.8 | **6857.0** | 6279.3 |
| jamesbond | 302.8 | 29.0 | 47.4 | 125.3 | 112.3 | 301.6 | **471.0** |
| kangaroo | 3035.0 | 52.0 | 0.0 | 323.1 | 605.4 | 779.3 | **872.5** |
| krull | 2665.5 | 1598.0 | 1468 | **4539.9** | 3277.9 | 2851.5 | 4229.6 |
| kung_fu_master | 22736.3 | 258.5 | 0. | **17257.2** | 5722.2 | 14346.1 | 14307.8 |
| ms_pacman | 6951.6 | 307.3 | 67 | **1480.0** | 941.9 | 1204.1 | 1465.5 |
| Pong | 14.6 | -20.7 | -20.6 | **12.8** | 1.3 | -19.3 | -16.5 |
| Private eye | 69571.3 | 24.9 | 0 | 58.3 | 100.0 | 97.8 | **218.4** |
| Qbert | 13455.0 | 163.9 | 123.46 | **1288.8** | 509.3 | 1152.9 | 1042.4 |
| Road_Runner | 7845.0 | 11.5 | 1588.46 | 5640.6 | 2696.7 | **9600.0** | 5661.0 |
| seaquest | 42054.7 | 68.4 | 131.69 | **683.3** | 286.92 | 354.1 | 384.5 |
| Up_n_Down | 11693.2 | 533.4 | 504.6 | **3350.3** | 2847.6 | 2877.4 | 2955.2 |

[177] where the agent operates purely from pixels. The reason for operating in these environments is multi fold: (i) they present a reasonably challenging and diverse set of tasks; (ii) sample-efficiency of pure model-free RL algorithms operating from pixels on these benchmarks is poor; (iii) multiple recent efforts to improve the sample efficiency of both model-free and model-based methods on

these benchmarks thereby giving us sufficient baselines to compare against; (iv) performance on the DM control suite is relevant to robot learning in real world benchmarks.

We run experiments on sixteen environments from DMControl to examine the performance of CURL on pixels relative to SAC with access to the ground truth state, shown in Figure 3.7. For more extensive benchmarking, we compare CURL to five leading pixel-based methods across the the six environments presented in [202]: ball-in-cup, finger-spin, reacher-easy, cheetah-run, walker-walk, cartpole-swingup for benchmarking.

**Atari:** Similar to DMControl sample-efficiency benchmarks, there have been a number of recent papers that have benchmarked for sample-efficiency on the Atari 2600 Games. [95] proposed comparing various algorithms in terms of performance achieved within $100$K timesteps ($400$K frames, frame skip of $4$) of interaction with the environments (games). The method proposed by [95] called SimPLe is a model-based RL algorithm. SimPLe is compared to a random agent, model-free Rainbow DQN [81] and human performance for the same amount of interaction time. Recently, [72] and [98] proposed data-efficient versions of Rainbow DQN which are competitive with SimPLe on the same benchmark. Given that the same benchmark has been established in multiple recent papers and that there is a human baseline to compare to, we benchmark CURL on all the 26 Atari Games (Table 3.2).

## Baselines for benchmarking sample efficiency

**DMControl baselines:** We present a number of baselines for continuous control within the DM-Control suite: (i) SAC-AE [202] where the authors attempt to use a $\beta$-VAE [82], VAE [100] and a regualrized autoencoder [187, 56] jointly with SAC; (ii) SLAC [114] which learns a latent space world model on top of VAE features [66] and builds value functions on top; (iii) PlaNet and (iv) Dreamer [70, 69] both of which learn a latent space world model and explicitly plan through it; (v) Pixel SAC: Vanilla SAC operating purely from pixels [67]. These baselines are competitive methods for benchmarking control from pixels. In addition to these, we also present the baseline State-SAC where the assumption is that the agent has access to low level state based features and does not operate from pixels. This baseline acts as an oracle in that it approximates the upper bound of how sample-efficient a pixel-based agent can get in these environments.

**Atari baselines**: For benchmarking performance on Atari, we compare CURL to (i) SimPLe [95], the top performing model-based method in terms of data-efficiency on Atari and (ii) Rainbow DQN [81], a top-performing model-free baseline for Atari, (iii) OTRainbow [98] which is an Over-Trained version of Rainbow for data-efficiency, (iv) Efficient Rainbow [72] which is a modification of Rainbow hyperparameters for data-efficiency, (v) Random Agent [95], (vi) Human Performance [95, 72]. All the baselines and our method are evaluated for performance after 100K interaction steps (400K frames with a frame skip of 4) which corresponds to roughly two hours of gameplay. These benchmarks help us understand how the state-of-the-art pixel based RL algorithms compare in terms of sample efficiency and also to human efficiency. **Note:** Scores for SimPLe and Human baselines have been reported differently in prior work [98, 72]. To be rigorous, we take the best reported score for each individual game reported in prior work.

## 3.6 Results

### DMControl

Sample-efficiency results for DMControl experiments are shown in Table 3.1 and in Figures 3.4, 3.6, and 3.7. Below are the key findings:

(i) CURL is the **state-of-the-art image-based RL algorithm** on the majority (**5** out of **6**) DMControl environments that we benchmark on for sample-efficiency against existing pixel-based baselines. On DMControl100k, CURL achieves **1.9x** higher median performance than Dreamer [69], a leading model-based method, and is **4.5x** more data-efficient shown in Figure 3.6.

(ii) CURL operating purely from pixels **nearly matches** (and sometimes surpasses) **the sample efficiency of SAC operating from state** on the majority of 16 DMControl environments tested shown in Figure 3.7 and matches the median state-based score on DMControl500k shown in Figure 3.4. This is a **first** for any image-based RL algorithm, be it model-based, model-free, with or without auxiliary tasks.

(iii) CURL solves (converges close to optimal score of 1000) on the majority of 16 DMControl experiments within **500k** steps. It also matches the state-based median score across the 6 extensively benchmarked environments in this regime.

### Atari

Results for Atari100k are shown in Table 3.2. Below are the key findings:

(i) CURL achieves a median human-normalized score (HNS) of **17.5%** while SimPLe and Efficient Rainbow DQN achieve 14.4% and 16.1% respectively. The mean HNS is 38.1%, 44.3%, and 28.5% for CURL, SimPLe, and Efficient Rainbow DQN respectively.

(ii) CURL improves on top of Efficient Rainbow on **19** out of **26** Atari games. Averaged across 26 games, CURL improves on top of Efficient Rainbow by **1.3x**, while the median performance improvement over SimPLE and Efficient Rainbow are **1.2x** and 1.1x respectively.

(iii) CURL **surpasses human performance** on two games JamesBond (1.6 HNS), Krull (2.5 HNS).

## 3.7 Ablation Studies

In Appendix 3.12, we present the results of ablation studies carried out to answer the following questions: (i) Does CURL learn only visual features or does it also capture temporal dynamics of the environment? (ii) How well does the RL policy perform if CURL representations are learned solely with the contrastive objective and no signal from RL? (iii) Why does CURL match state-based RL performance on some DMControl environments but not on others?

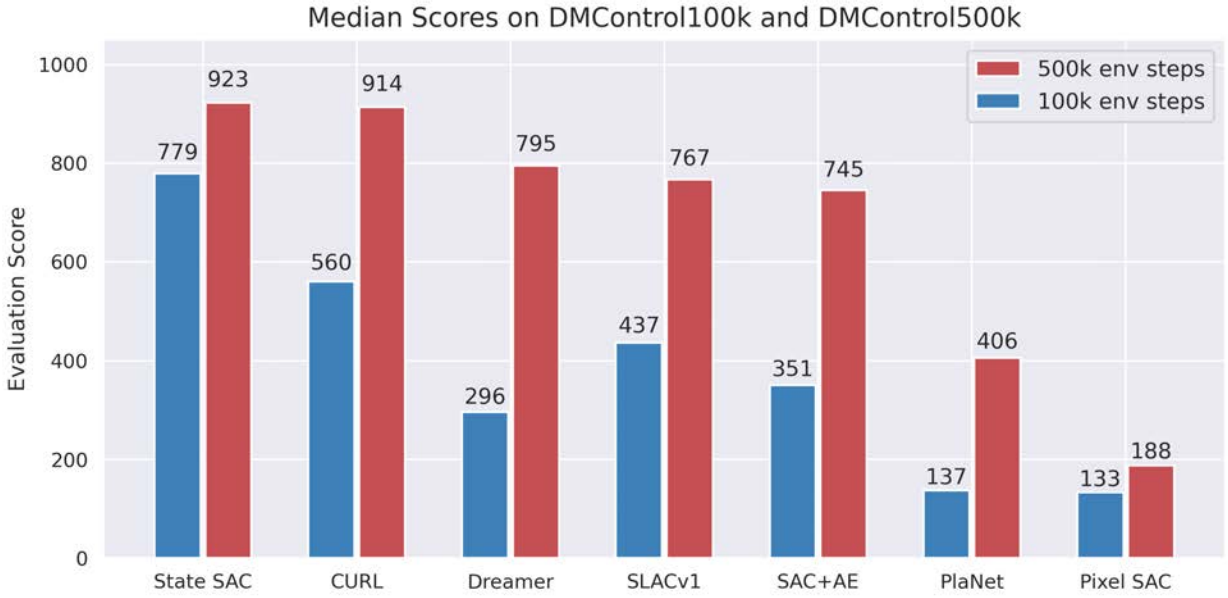



Figure 3.4: Performance of CURL coupled to SAC averaged across 10 seeds relative to SLACv1, PlaNet, Pixel SAC and State SAC baselines. At the 500k benchmark CURL matches the median score of state-based SAC. At 100k environment steps CURL achieves a 1.9x higher median score than Dreamer. For a direct comparison, we only compute the median across the 6 environments in 3.1 (4 for SLAC) and show learning curves for CURL across 16 DMControl experiments in 3.7.

## 3.8 Implementation Details

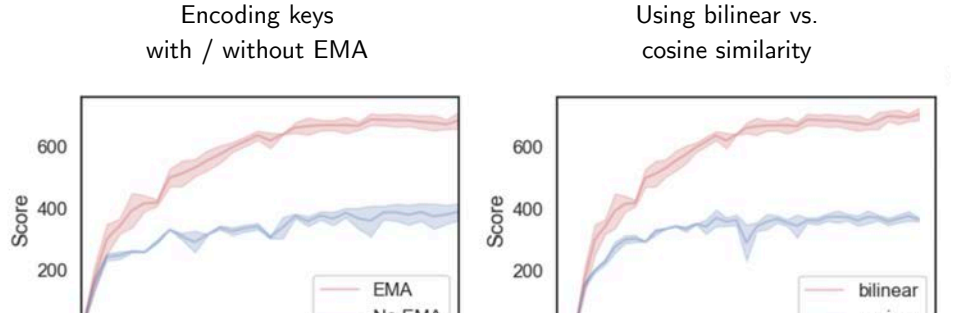
Below, we explain the implementation details for CURL in the DMControl setting. Specifically, we use the SAC algorithm as the RL objective coupled with CURL and build on top of the publicly released implementation from [202]. We present in detail the hyperparameters for the architecture and optimization. We do not use any extra hyperparameter for balancing the contrastive loss and the reinforcement learning losses. Both the objectives are weighed equally in the gradient updates.

**Architecture:** We use an encoder architecture that is similar to [202], which we sketch in PyTorch-like pseudocode below. The actor and critic both use the same encoder to embed image observations. A full list of hyperparameters is displayed in Table 3.3.

For contrastive learning, CURL utilizes momentum for the key encoder [77] and a bi-linear inner product as the similarity measure [135]. Performance curves ablating these two architectural choices are shown in Figure 3.5.

Table 3.3: Hyperparameters used for DMControl CURL experiments. Most hyperparameters values are unchanged across environments with the exception for action repeat, learning rate, and batch size.

| **Hyperparameter** | **Value** |
|---|---|
| Random crop | True |
| Observation rendering | $(100, 100)$ |
| Observation downsampling | $(84, 84)$ |
| Replay buffer size | 100000 |
| Initial steps | 1000 |
| Stacked frames | 3 |
| Action repeat | 2 finger, spin; walker, walk |
| | 8 cartpole, swingup |
| | 4 otherwise |
| Hidden units (MLP) | 1024 |
| Evaluation episodes | 10 |
| Optimizer | Adam |
| $(\beta_1, \beta_2) \rightarrow (f_\theta, \pi_\psi, Q_\phi)$ | $(.9, .999)$ |
| $(\beta_1, \beta_2) \rightarrow (\alpha)$ | $(.5, .999)$ |
| Learning rate $(f_\theta, \pi_\psi, Q_\phi)$ | $2e-4$ cheetah, run |
| | $1e-3$ otherwise |
| Learning rate $(\alpha)$ | $1e-4$ |
| Batch Size | 512 |
| $Q$ function EMA $\tau$ | 0.01 |
| Critic target update freq | 2 |
| Convolutional layers | 4 |
| Number of filters | 32 |
| Non-linearity | ReLU |
| Encoder EMA $\tau$ | 0.05 |
| Latent dimension | 50 |
| Discount $\gamma$ | .99 |
| Initial temperature | 0.1 |



Pseudo-code for the architecture is provided below:

```
def encode(x,z_dim):
  """
     ConvNet encoder
     args:
         B-batch_size, C-channels
         H,W-spatial_dims
         x : shape : [B, C, H, W]
         C = 3 * num_frames; 3 - R/G/B
         z_dim: latent dimension
     """

  x = x / 255.

  # c: channels, f: filters
  # k: kernel, s: stride

  z = Conv2d(c=x.shape[1], f=32, k=3, s=2)])(x)
  z = ReLU(z)

  for _ in range(num_layers - 1):
    z = Conv2d((c=32, f=32, k=3, s=1))(z)
    z = ReLU(z)

  z = flatten(z)

  # in: input dim, out: output_dim, h: hiddens

  z = mlp(in=z.size(),out=z_dim,h=1024)
  z = LayerNorm(z)
  z = tanh(z)
```

**Terminology**: A common point of confusion is the meaning “training steps." We use the term *environment steps* to denote the amount of times the simulator environment is stepped through and *interaction steps* to denote the number of times the agent steps through its policy. The terms *action repeat* or *frame skip* refer to the number of times an action is repeated when it’s drawn from the agent’s policy. For example, if action repeat is set to 4, then 100k interaction steps is equivalent to 400k environment steps.

**Batch Updates:** After initializing the replay buffer with observations extracted by a random agent, we sample a batch of observations, compute the CURL objectives, and step through the optimizer. Note that since queries and keys are generated by data-augmenting an observation, we can generate arbitrarily many keys to increase the contrastive batch size without sampling any additional observations.

**Shared Representations:** The objective of performing contrastive learning together with RL is to ensure that the shared encoder learns rich features that facilitate sample efficient control. There is a subtle coincidental connection between MoCo and off-policy RL. Both the frameworks adopt the usage of a momentum averaged (EMA) version of the underlying model. In MoCo, the EMA encoder is used for encoding the keys (targets) while in off-policy RL, the EMA version of the Q-networks are used as targets in the Bellman error [133, 67]. Thanks to this connection, CURL shares the convolutional encoder, momentum coefficient and EMA update between contrastive and reinforcement learning updates for the shared parameters. The MLP part of the critic that operates

on top of these convolutional features has a separate momentum coefficient and update decoupled from the image encoder parameters.

**Balancing Contrastive and RL Updates**: While past work has learned hyperparameters to balance the auxiliary loss coefficient or learning rate relative to the RL objective [92, 202], CURL does not need any such adjustments. We use both the contrastive and RL objectives together with equal weight and learning rate. This simplifies the training process compared to other methods, such as training a VAE jointly [70, 69, 114], that require careful tuning of coefficients for representation learning.

**Differences in Data Collection between Computer Vision and RL Settings**: There are two key differences between contrastive learning in the computer vision and RL settings because of their different goals. Unsupervised feature learning methods built for downstream vision tasks like image classification assume a setting where there is a large static dataset of unlabeled images. On the other hand, in RL, the dataset changes over time to account for the agent's new experiences. Secondly, the size of the memory bank of labeled images and dataset of unlabeled ones in vision-based settings are 65K and 1M (or 1B) respectively. The goal in vision-based methods is to learn from millions of unlabeled images. On the other hand, the goal in CURL is to develop sample-efficient RL algorithms. For example, to be able to solve a task within 100K timesteps (approximately 2 hours in real-time), an agent can only ingest 100K image frames.

Therefore, unlike MoCo, CURL does not use a memory bank for contrastive learning. Instead, the negatives are constructed on the fly for every minibatch sampled from the agent's replay buffer for an RL update similar to SimCLR. The exact implementation is provided as a PyTorch-like code snippet in 3.4.

**Data Augmentation:**

Random crop data augmentation has been crucial for the performance of deep learning based computer vision systems in object recognition, detection and segmentation [104, 174, 34, 26]. However, similar augmentation methods have not seen much adoption in the field of RL even though several benchmarks use raw pixels as inputs to the model.

CURL adopts the random crop data augmentation as the stochastic data augmentation applied to a frame stack. To make it easier for the model to correlate spatio-temporal patterns in the input, we apply the same random crop (in terms of box coordinates) across all four frames in the stack as opposed to extracting different random crop positions from each frame in the stack. Further, unlike in computer vision systems where the aspect ratio for random crop is allowed to be as low as 0.08, we preserve much of the spatial information as possible and use a constant aspect ratio of 0.84 between the original and cropped. In our experiments, data augmented samples for CURL are formed by cropping $84 \times 84$ frames from an input frame of $100 \times 100$.

**DMControl:** We render observations at $100 \times 100$ and randomly crop $84 \times 84$ frames. For evaluation, we render observations at $100 \times 100$ and center crop to $84 \times 84$ pixels. We found that implementing random crop efficiently was extremely important to the success of the algorithm. We provide pseudocode below:

```
from skimage import view_as_windows
import numpy as np

```

```
def random_crop(imgs, out):
  """
     Vectorized random crop
     args:
         imgs: shape (B,C,H,W)
         out: output size (e.g. 84)
     """

  # n: batch size.
  n = imgs.shape[0]
  img_size = imgs.shape[-1] # e.g. 100
  crop_max = img_size - out

  imgs = np.transpose(imgs, (0, 2, 3, 1))

  w1 = np.random.randint(0, crop_max, n)
  h1 = np.random.randint(0, crop_max, n)

  # creates all sliding window
  # combinations of size (out)

  windows = view_as_windows(
    imgs, (1, out, out, 1))[..., 0,:,:, 0]

  # selects a random window
  # for each batch element
  cropped = windows[np.arange(n), w1, h1]
  return cropped
```

## 3.9 Atari100k Implementation Details

The flexibility of CURL allows us to apply it to discrete control setting with minimal modifications. Similar to our rationale for picking SAC as the baseline RL algorithm to couple CURL with (for continuous control), we pick the data-efficient version of Rainbow DQN (Efficient Rainbow) [72] for Atari100K which performs competitively with an older version of SimPLe (most recent version has improved numbers). In order to understand specifically what the gains from CURL are without any other changes, we adopt the exact same hyperparameters specified in the paper [72] (including a modified convolutional encoder that uses larger kernel size and stride of 5). We present the details in Table 3.4. Similar to DMControl, the contrastive objective and the RL objective are weighted equally for learning (except for Pong, Freeway, Boxing and PrivateEye for which we used a coefficient of $0.05$ for the momentum contastive loss. On a large majority (22 out of 26) of the games, we do not use this adjustment. While it is standard practice to use the same hyperparameters for all games in Atari, papers proposing auxiliary losses have adopted a different practice of using game specific coefficients [92].). We use the Efficient Rainbow codebase from `https://github.com/Kaixhin/Rainbow` which has a reproduced version of [72]. We evaluate with $20$ random seeds and report the mean score for each game given the high variance nature of the Atari100k steps benchmark. We restrict ourselves to using grayscale renderings of image observations and use random crop of frame stack as data augmentation.

Table 3.4: Hyperparameters used for Atari100K CURL experiments. Hyperparameters are unchanged across games.

| **Hyperparameter** | **Value** |
|---|---|
| Random crop | True |
| Image size | $(84, 84)$ |
| Data Augmentation | Random Crop (Train) |
| Replay buffer size | 100000 |
| Training frames | 400000 |
| Training steps | 100000 |
| Frame skip | 4 |
| Stacked frames | 4 |
| Action repeat | 4 |
| Replay period every | 1 |
| Q network: channels | 32, 64 |
| Q network: filter size | $5 \times 5, 5 \times 5$ |
| Q network: stride | 5, 5 |
| Q network: hidden units | 256 |
| Momentum (EMA for CURL) $\tau$ | 0.001 |
| Non-linearity | ReLU |
| Reward Clipping | $[-1, 1]$ |
| Multi step return | 20 |
| Minimum replay size for sampling | 1600 |
| Max frames per episode | 108K |
| Update | Distributional Double Q |
| Target Network Update Period | every 2000 updates |
| Support-of-Q-distribution | 51 bins |
| Discount $\gamma$ | 0.99 |
| Batch Size | 32 |
| Optimizer | Adam |
| Optimizer: learning rate | 0.0001 |
| Optimizer: $\beta 1$ | 0.9 |
| Optimizer: $\beta 2$ | 0.999 |
| Optimizer $\epsilon$ | 0.000015 |
| Max gradient norm | 10 |
| Exploration | Noisy Nets |
| Noisy nets parameter | 0.1 |
| Priority exponent | 0.5 |
| Priority correction | $0.4 \rightarrow 1$ |
| Hardware | CPU |

## 3.10 Benchmarking Data Efficiency

Tables 3.1 and 3.2 show the episode returns of DMControl100k, DMControl500k, and Atari100k across CURL and a number of pixel-based baselines. CURL outperforms all baseline pixel-based methods across experiments on both DMControl100k and DMControl500k. On Atari100k experiments, CURL coupled with Eff Rainbow outperforms the baseline on the majority of games tested (19 out of 26 games).

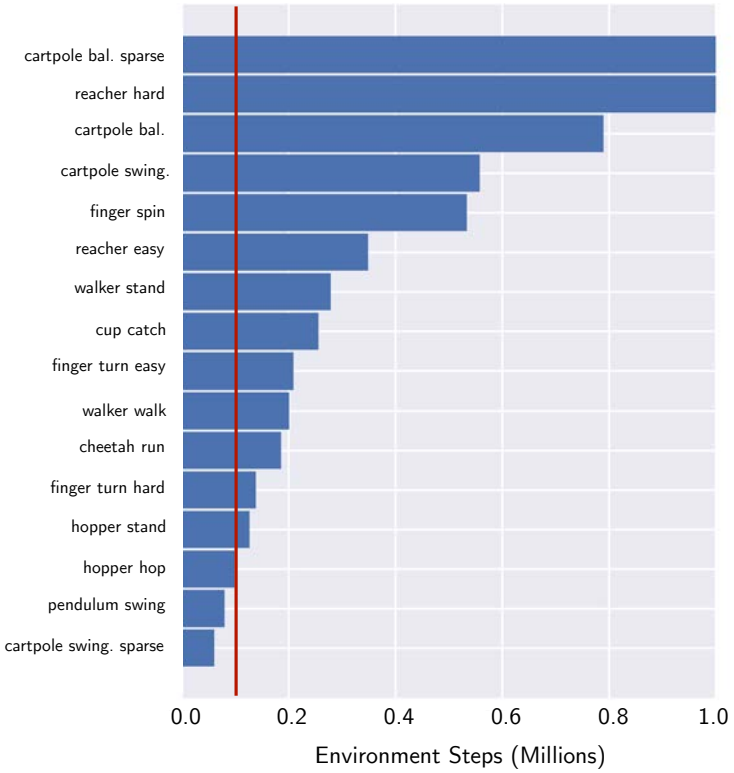



Figure 3.6: The number of steps it takes a prior leading pixel-based method, Dreamer, to achieve the same score that CURL achieves at 100k training steps (clipped at 1M steps). On average, CURL is 4.5x more data-efficient. We chose Dreamer because the authors [69] report performance for all of the above environments while other baselines like SLAC and SAC+AE only benchmark on 4 and 6 environments, respectively. For further comparison of CURL with these methods, the reader is referred to Table 3.1 and Figure 3.4.

## 3.11 Further Investigation of Data-Efficiency in Contrastive RL

To further benchmark CURL's sample-efficiency, we compare it to state-based SAC on a total of 16 DMControl environments. Shown in Figure 3.7, CURL matches state-based data-efficiency on most of the environments, but lags behind state-based SAC on more challenging environments.

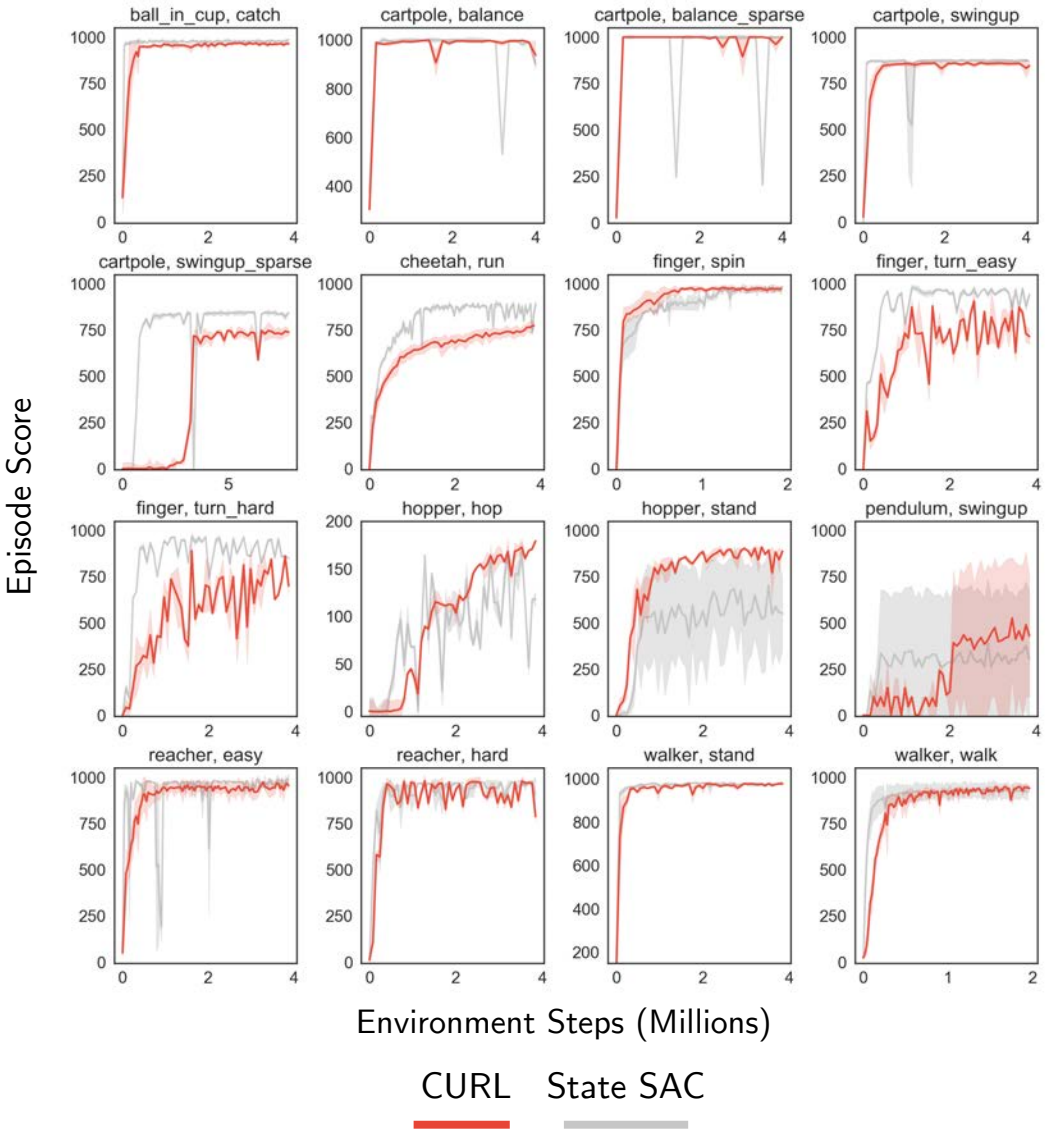



Figure 3.7: CURL compared to state-based SAC run for 3 seeds on each of 16 selected DMControl environments. For the 6 environments in 3.4, CURL performance is averaged over 10 seeds.

## 3.12 Ablations

### Learning Temporal Dynamics

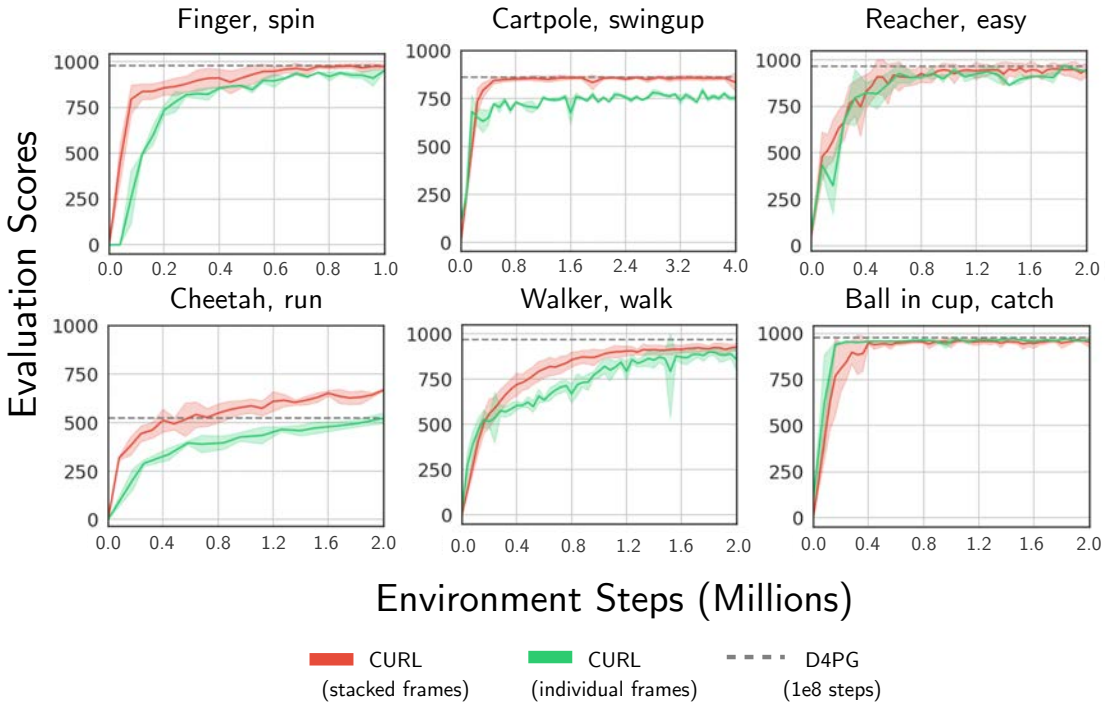



Figure 3.8: CURL with temporal and visual discrimination (red) compared to CURL with only visual discrimination (green). In most settings, the variant with temporal variant outperforms the purely visual variant of CURL. The two exceptions are reacher and ball in cup environments, suggesting that learning dynamics is not necessary for those two environments. Note that the walker environment was run with action repeat of 4, whereas walker walk in the main results Table 3.1 and Figure 3.7 was run with action repeat of 2.

To gain insight as to whether CURL learns temporal dynamics across the stacked frames, we also train a variant of CURL where the discriminants are individual frames as opposed to stacked ones. This can be done by sampling stacked frames from the replay buffer but only using the first frame to update the contrastive loss:

```
f_q = x_q[:,:3,...] # (B,C,H,W), C=9.
f_k = x_k[:,:3,...]
```

During the actor-critic update, frames in the batch are encoded individually into latent codes, which are then concatenated before being passed to a dense network.

```
# x: (B,C,H,W), C=9.
z1 = encode(x[:,:3,...])
z2 = encode(x[:,3:6,...])
z3 = encode(x[:,6:9,...])
z = torch.cat([z1,z2,z3],-1)
```

Encoding each frame indiviudally ensures that the contrastive objective only has access to visual discriminants. Comparing the visual and spatiotemporal variants of CURL in Figure 3.8 shows that the variant trained on stacked frames outperforms the visual-only version in most environments. The only exceptions are reacher and ball-in-cup environments. Indeed, in those environments the visual signal is strong enough to solve the task optimally, whereas in other environments, such as walker and cheetah, where balance or coordination is required, visual information alone is insufficient.

## Increasing Gradient Updates per Agent Step

Although most baselines we benchmark against use one gradient update per agent step, it was recently empirically shown that increasing the ratio of gradients per step improves data-efficiency in RL [98]. This finding is also supported by SLAC [114], where results are shown with a ratio of 1:1 (SLACv1) and 3:1 (SLACv2). We

Table 3.5: Scores achieved by CURL and SLAC when run with a 3:1 ratio of gradient updates per agent step on DMControl500k and DMControl100k. CURL achieves state-of-the-art performance on the majority (**3** out of **4**) environments on DMControl500k. Performance of both algorithms is improved relative to the 1:1 ratio reported for all baselines in Table 3.1 but at the cost of significant compute and wall-clock time overhead.

| DMControl500k | CURL | SLACv2 |
|---|---|---|
| Finger, spin | **923** $\pm$ **50** | 884 $\pm$ 98 |
| Walker, walk | **911** $\pm$ **35** | 891 $\pm$ 60 |
| Cheetah, run | 545 $\pm$ 39 | **791** $\pm$ **37** |
| Ball in cup, catch | **948** $\pm$ **21** | 885 $\pm$ 154 |
| DMControl100k | CURL | SLACv2 |
| Finger, spin | **741**$\pm$ **118** | 728 $\pm$212 |
| Walker, walk | 428 $\pm$ 59 | **513** $\pm$ **41** |
| Cheetah, run | 314 $\pm$ 46 | **438** $\pm$ **76** |
| Ball in cup, catch | **899** $\pm$ **47** | 837 $\pm$ 147 |

## Decoupling Representation Learning from Reinforcement Learning

Typically, Deep RL representations depend almost entirely on the reward function specific to a task. However, hand-crafted representations such as the proprioceptive state are independent of the reward function. It is much more desirable to learn reward-agnostic representations, so that the same representation can be re-used across different RL tasks. We test whether CURL can learn such representations by comparing CURL to a variant where the critic gradients are backpropagated through the critic and contrastive dense feedforward networks but stopped before reaching the convolutional neural network (CNN) part of the encoder.

Scores displayed in Figure 3.9 show that for many environments, the detached CNN representations are sufficient to learn an optimal policy. The major exception is the cheetah environment, where the detached representation significantly under-performs. Though promising, we leave further exploration of task-agnostic representations for future work.

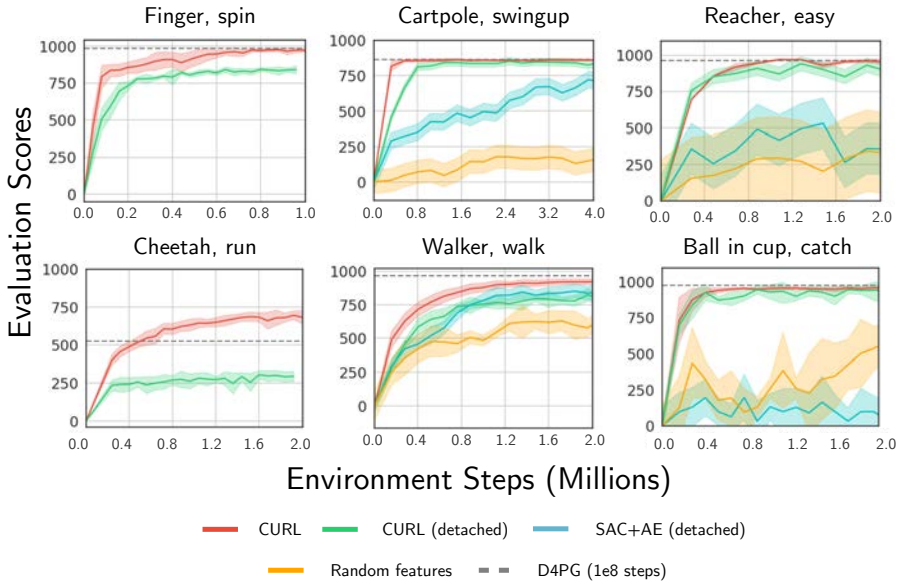



Figure 3.9: CURL where the CNN part of the encoder receives gradients from both the contrastive loss and critic (red) compared to CURL with the convolutional part of the encoder trained only with the contrastive objective (green). The detached encoder variant is able to learn representations that enable near-optimal learning on most environments, except for cheetah. As in Figure 3.8, the walker environment was run with action repeat of 4.

## Removing Data Augmentation for the Actor Critic

Our main results involve the use of data augmentations to regularize both the contrastive and SAC objectives. Here, we investigate whether the contrastive representations alone are sufficient for learning effective policies. In these experiments, we only augment the data for the contrastive objective but not for the SAC agent. As a result, data augmentation is used only to learn features but does not influence the control policy. The pseudocode is shown below:

```
# o = original unaugmented observation
# aug = augmentation
# contrastive = InfoNCE loss
o_anchor, o_target = aug(o), aug(o)
curl_loss = contrastive(o_anchor, o_target)
sac_loss = critic_loss(o) + actor_loss(o)
loss = curl_loss + sac_loss
params = update(params, grad(loss, params))
```

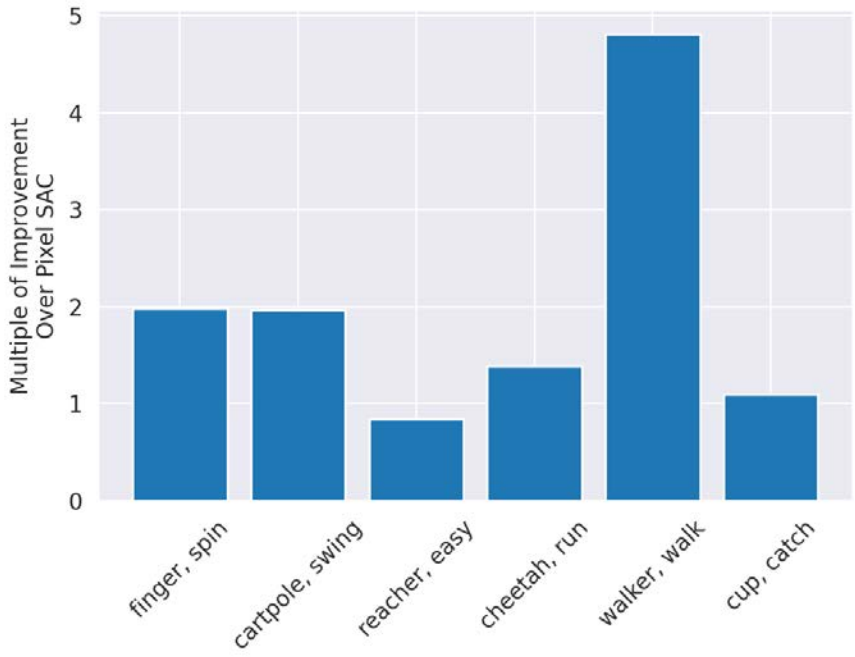



Figure 3.10: CURL with no data augmentations passed to the SAC agent improves the performance of the baseline pixel SAC by a mean of 2.0x / median of 1.7x on DMControl500k. For these runs we use a smaller batch size of 128 than the 512 batch size used for results in Table 3.4. While the constastive loss alone improves over the pixel SAC baseline, most environments benefit from data augmentation also being passed to the SAC agent.

DMControl500k results plotted in Figure 3.10 show that, on average, features learned through the contrastive loss alone improve the pixel SAC baseline by 2x. Augmenting the input passed to

the SAC algorithm further improves performance.

## Predicting State from Pixels

Despite improved sample-efficiency on most DMControl tasks, there is still a visible gap between the performance of SAC on state and SAC with CURL in some environments. Since CURL learns representations by performing instance discrimination across stacks of three frames, it's possible that the reason for degraded sample-efficiency on more challenging tasks is due to partial-observability of the ground truth state.

To test this hypothesis, we perform supervised regression $(X, Y)$ from pixels $X$ to the proprioceptive state $Y$, where each data point $x \in X$ is a stack of three consecutive frames and $y \in Y$ is the corresponding state extracted from the simulator. We find that the error in predicting the state from pixels correlates with the policy performance of pixel-based methods. Test-time error rates displayed in Figure 3.11 show that environments that CURL solves as efficiently as state-based SAC have low error-rates in predicting the state from stacks of pixels. The prediction error increases for more challenging environments, such as cheetah-run and walker-walk. Finally, the error is highest for environments where current pixel-based methods, CURL included, make no progress at all [177], such as humanoid and swimmer.

This investigation suggests that degraded policy performance on challenging tasks may result from the lack of requisite information about the underlying state in the pixel data used for learning representations. We leave further investigation for future work.

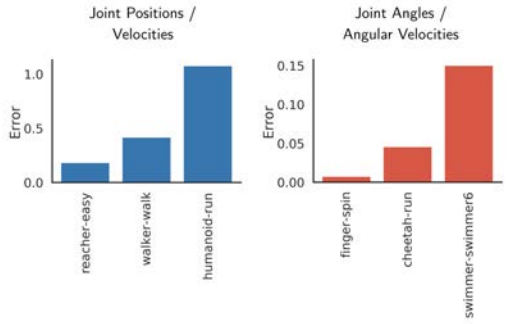



Figure 3.11: Test-time mean squared error for predicting the proprioceptive state from pixels on a number of DMControl environments. In DMControl, environments fall into two groups - where the state corresponds to either (a) positions and velocities of the robot joints or (b) the joint angles and angular velocities.

## CURL + Efficient Rainbow Atari runs

We report the scores (Tables 3.6 and 3.7) for 20 seeds across the 26 Atari games in the Atari100k benchmark for CURL coupled with Efficient Rainbow. The variance across multiple seeds is

considerably high in this benchmark. Therefore, we report the scores for each of the seeds along with the mean and standard deviation for each game.

| Pacman | Fbite | Asterix | KungFu | Kngroo | Gopher | RRunner | JBond | BZone | Cquest | Assault | Krull | Qbert |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1287 | 2292 | 850 | 8470 | 600 | 1036 | 2820 | 305 | 18100 | 322 | 634.2 | 3404.3 | 1020 |
| 1608 | 1046 | 525 | 10870 | 2280 | 574 | 3190 | 265 | 18200 | 236 | 696.8 | 2443.5 | 650 |
| 1466 | 1209 | 655 | 10920 | 1940 | 540 | 7840 | 335 | 26800 | 352 | 655.2 | 6791.4 | 830 |
| 1430 | 255 | 565 | 7730 | 1140 | 618 | 12060 | 145 | 21300 | 386 | 443 | 3022.5 | 902.5 |
| 1114 | 426 | 715 | 17525 | 520 | 534 | 8340 | 565 | 7900 | 458 | 546 | 3892.2 | 3957.5 |
| 1083 | 2280 | 715 | 3560 | 600 | 596 | 6920 | 565 | 8100 | 224 | 564.9 | 3505.5 | 772.5 |
| 2301 | 259 | 770 | 10940 | 600 | 502 | 2230 | 350 | 12000 | 282 | 514.4 | 2564.1 | 782.5 |
| 1128 | 335 | 980 | 23420 | 900 | 998 | 4250 | 365 | 16500 | 339 | 516.6 | 4079.7 | 727.5 |
| 1184 | 1409 | 665 | 15160 | 600 | 950 | 1570 | 140 | 23900 | 526 | 661.5 | 2376.4 | 705 |
| 1510 | 258 | 610 | 15370 | 730 | 544 | 6300 | 425 | 19900 | 436 | 664.5 | 4161.8 | 757.5 |
| 2343 | 335 | 905 | 22260 | 600 | 796 | 3100 | 315 | 10000 | 272 | 529 | 3311.1 | 647.5 |
| 1063 | 1062 | 800 | 17320 | 880 | 522 | 1060 | 335 | 11200 | 428 | 445.2 | 2517.3 | 562.5 |
| 2040 | 1542 | 675 | 31820 | 220 | 392 | 6050 | 735 | 9700 | 358 | 573.3 | 3764.7 | 2425 |
| 1195 | 1102 | 795 | 23360 | 920 | 780 | 11810 | 950 | 23500 | 533 | 531.3 | 10150.2 | 1112.5 |
| 1343 | 2461 | 585 | 27460 | 600 | 792 | 4630 | 520 | 10500 | 968 | 663.6 | 2883.6 | 527.5 |
| 1354 | 257 | 865 | 7770 | 2300 | 454 | 2530 | 755 | 18100 | 314 | 795.3 | 5123.7 | 472.5 |
| 1925 | 513 | 730 | 8820 | 320 | 564 | 6840 | 750 | 9000 | 378 | 633 | 3652.5 | 610 |
| 1228 | 1826 | 680 | 2980 | 600 | 522 | 6580 | 795 | 8900 | 168 | 674.1 | 2376.4 | 697.5 |
| 1099 | 1889 | 965 | 10100 | 600 | 496 | 10720 | 450 | 10700 | 242 | 604.8 | 11745 | 1847.5 |
| 1608 | 2869 | 640 | 10300 | 500 | 1176 | 4380 | 355 | 13100 | 467 | 665.7 | 2826 | 840 |
| 1465.5 | 1181.3 | 734.5 | 14307.8 | 872.5 | 669.3 | 5661 | 471 | 14870 | 384.5 | 600.6 | 4229.6 | 1042.4 |
| 397.5 | 856.2 | 129.8 | 7919.3 | 600.1 | 220.6 | 3289.3 | 226.2 | 5964.3 | 170.2 | 89.5 | 2540.6 | 828.4 |

Table 3.6: CURL implemented on top of Efficient Rainbow - Scores reported for 20 random seeds for each of the above games, with the last two rows being the mean and standard deviation across the runs.

## 3.13 Connection to work on data augmentations

Recently, there have been two papers published on using data augmentations for reinforcement learning, RAD [110] and DrQ [103]. These two papers present the version of CURL without an auxiliary contrastive loss but rather directly feeding in the augmented views of the image observations to the underlying value / policy network(s). Both RAD and DrQ present results on both continuous and discrete control environments, surpassing the results presented in CURL on both the DMControl and Atari benchmarks. Plenty of researchers have opined in public forums whether the results in RAD and DrQ make CURL irrelevant if the objective is to use data augmentations for data-efficient reinforcement learning. We believe that answering this question needs more nuance and present our opinions below:

| UDown | Hero | Climber | Chopper | DAttack | Amidar | Alien | BHeist | Breakout | Freeway | Pong | PrEye | Boxing |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 3529 | 8747.5 | 19090 | 560 | 611.5 | 150.9 | 616 | 95 | 3.6 | 29.2 | -19.3 | 100 | -0.5 |
| 772 | 3026 | 8290 | 1530 | 707.5 | 131.2 | 923 | 184 | 5 | 25.4 | -16.9 | 100 | -11.4 |
| 5972 | 7146 | 12160 | 1390 | 843.5 | 141.5 | 467 | 75 | 3.2 | 27.6 | -12 | 100 | 4 |
| 2793 | 7686 | 8920 | 1100 | 330.5 | 133.7 | 441 | 232 | 5.1 | 28.6 | -19.6 | 100 | 3.6 |
| 3546 | 7335 | 11360 | 500 | 759 | 157.1 | 716 | 187 | 2.9 | 22.8 | -17.8 | 1357.4 | 6.2 |
| 4552 | 7325 | 4110 | 990 | 940 | 125.4 | 453 | 367 | 6.3 | 29.6 | -18.9 | 100 | 5 |
| 2972 | 7275.5 | 9460 | 780 | 1136 | 183.2 | 273 | 186 | 5.9 | 23.3 | -15.9 | 0 | -1.7 |
| 2865 | 3115 | 20630 | 1180 | 758 | 153.6 | 540 | 68 | 2.6 | 27.6 | -15.2 | 100 | 0.1 |
| 3098 | 7424 | 6780 | 1380 | 772.5 | 127.8 | 499 | 60 | 5.9 | 26.1 | -18.7 | 100 | 3.5 |
| 1953 | 7475 | 13570 | 970 | 820 | 149.4 | 475 | 123 | 4.3 | 28.3 | -13.3 | 100 | -0.5 |
| 1467 | 3135 | 11890 | 1200 | 784 | 125.7 | 553 | 72 | 3.2 | 21.8 | -17.2 | 1510 | -22.1 |
| 2912 | 5060.5 | 9160 | 1130 | 1080 | 130.4 | 446 | 53 | 4.8 | 21.8 | -20.1 | 100 | -1.8 |
| 4123 | 4409 | 10960 | 1380 | 847 | 133 | 533 | 68 | 6.3 | 28.9 | -16.5 | 100 | 1.6 |
| 2334 | 6979 | 17360 | 1230 | 771.5 | 140.5 | 968 | 36 | 7.3 | 28.2 | -14.9 | 100 | 3.6 |
| 2605 | 4159 | 8930 | 1350 | 907.5 | 133.8 | 499 | 53 | 4.8 | 28.3 | -19.3 | 100 | -17.6 |
| 2432 | 7560 | 11510 | 1080 | 1095.5 | 191.8 | 523 | 105 | 3.7 | 26.8 | -15.6 | 0 | 21.7 |
| 3826 | 8587 | 22690 | 1210 | 700 | 115.5 | 616 | 276 | 6.6 | 27.5 | -21 | 100 | 2 |
| 3052 | 4683.5 | 8120 | 840 | 803.5 | 164 | 475 | 69 | 5.5 | 26.5 | -10.5 | 0 | 5.9 |
| 3131 | 7317 | 13500 | 730 | 818 | 131.7 | 525 | 50 | 4.3 | 26.8 | -13.3 | 100 | 18.7 |
| 1169 | 7141 | 14440 | 640 | 866 | 122.4 | 622 | 273 | 6.2 | 28.6 | -13.1 | 100 | 3.7 |
| 2955.2 | 6279.3 | 12146.5 | 1058.5 | 817.6 | 142.1 | 558.2 | 131.6 | 4.9 | 26.7 | -16.5 | 218.4 | 1.2 |
| 1181.1 | 1871.5 | 4765.6 | 299.1 | 176.6 | 20.0 | 160.3 | 94.4 | 1.4 | 2.4 | 2.9 | 417.9 | 10.0 |

Table 3.7: CURL implemented on top of Efficient Rainbow - Scores reported for 20 random seeds for each of the above games, with the last two rows being the mean and standard deviation across the runs.

1. If one has access to a rich stream of rewards from the underlying environment and is interested in optimzing the performance in terms of average reward, RAD and DrQ are likely to work better than CURL. The reason for this is simply that RAD and DrQ directly optimize for the objective one cares about, while CURL introduces an additional auxiliary consistency objective.

2. If one does not have access to a rich stream of rewards and is interested in learning good latent spaces in a task agnostic manner that can allow for data-efficient controllers across multiple tasks, CURL is the only option since the contrastive objective in CURL is reward independent. Our ablation on the detached encoder with the CURL objective present evidence that one could build simple MLPs on top of the CURL features without fine-tuning the underlying encoder and still be data-efficient on many of the DMControl tasks.

3. Future work in data-efficient reinforcement learning, particularly for real world settings, is likely to require approaches that do not rely on reward functions. In such scenarios, CURL is likely to be the more preferred approach. Further, one could potentially use CURL in a scenario where unsupervised pre-training without reward functions is initially performed before fine-tuning to the RL objective across multiple tasks.

Given the above reasons, there isn't a straightforward answer as to which is the better algorithm and the answer really depends on what the researcher / practioner wants to solve. We also emphasize that CURL was the first approach that used data augmentations effectively to significantly improve the data-efficiency of model-free reinforcement learning methods with very simple changes and showed improvement over relatively more complex model-based methods. The augmentations and results in CURL inspired future work in the form of RAD and DrQ. We hope that the analysis and results presented in CURL encourage researchers to employ data augmentations, contrastive losses and unsupervised pre-training for future reinforcement learning research.

## 3.14 Conclusion

In this work, we proposed CURL, a contrastive unsupervised representation learning method for RL, that achieves state-of-the-art data-efficiency on pixel-based RL tasks across a diverse set of benchmark environments. CURL is the first model-free RL pipeline accelerated by contrastive learning with minimal architectural changes to demonstrate state-of-the-art performance on complex tasks so far dominated by approaches that have relied on learning world models and (or) decoder-based objectives. We hope that progress like CURL enables avenues for real-world deployment of RL in areas like robotics where data-efficiency is paramount.

## 3.15 Acknowledgements

This research is supported in part by DARPA through the Learning with Less Labels (LwLL) Program and by ONR through PECASE N000141612723. We also thank Wendy Shang for her help with Section 3.12; Zak Stone and Google TFRC for cloud credits; Danijar Hafner, Alex Lee, and Denis Yarats for sharing data for baselines; and Lerrel Pinto, Adam Stooke, Will Whitney, and Ankesh Anand for insightful discussions.

# Chapter 4

# Reinforcement Learning with Augmented Data

## 4.1 Introduction

Learning from visual observations is a fundamental problem in reinforcement learning (RL). Current success stories build on two key ideas: (a) using expressive convolutional neural networks (CNNs) [113] that provide strong spatial inductive bias; (b) better credit assignment [133, 118, 156] techniques that are crucial for sequential decision making. This combination of CNNs with modern RL algorithms has led to impressive success with human-level performance in Atari [133], super-human Go players [161], continuous control from pixels [118, 156] and learning policies for real-world robot grasping [96].

While these achievements are truly impressive, RL is notoriously plagued with poor data-efficiency and generalization capabilities [46, 80]. Real-world successes of reinforcement learning often require *months* of data-collection and (or) training [96, 2]. On the other hand, biological agents have the remarkable ability to learn quickly [107, 95], while being able to generalize to a wide variety of unseen tasks [53]. These challenges associated with RL are further exacerbated when we operate on pixels due to high-dimensional and partially-observable inputs. Bridging the gap of data-efficiency and generalization is hence pivotal to the real-world applicability of RL.

Supervised learning, in the context of computer vision, has addressed the problems of data-efficiency and generalization by injecting useful priors. One such often ignored prior is *Data Augmentation*. It was critical to the early successes of CNNs [113, 104] and has more recently enabled better supervised [33, 34], semi-supervised [200, 198, 16] and self-supervised [79, 26, 76] learning. By using multiple augmented views of the same data-point as input, CNNs are forced to learn consistencies in their internal representations. This results in a visual representation that improves generalization [198, 79, 26, 76], data-efficiency [200, 79, 26] and transfer learning [79, 76].

Inspired by the impact of data augmentation in computer vision, we present RAD: **R**einforcement Learning with **A**ugmented **D**ata, a technique to incorporate data augmentations on input observations

for reinforcement learning pipelines. Through RAD, we ensure that the agent is learning on multiple views (or augmentations) of the same input (see Figure 4.1). This allows the agent to improve on *two key capabilities*: (a) **data-efficiency**: learning to quickly master the task at hand with drastically fewer experience rollouts; (b) **generalization**: improving transfer to unseen tasks or levels simply by training on more diversely augmented samples. To the best of our knowledge, we present the first extensive study of the use of data augmentation for reinforcement learning with *no changes* to the underlying RL algorithm and no additional assumptions about the domain other than the knowledge that the agent operates from image-based or proprioceptive (positions & velocities) observations.

We highlight the main contributions of RAD below:

- We show that *RAD outperforms prior state-of-the-art baselines* on both the widely used pixel-based DeepMind control benchmark [177] as well as state-based OpenAI Gym benchmark [17]. On both benchmark, RAD sets a new state-of-the-art *in terms data-efficiency and asymptotic performance* on the majority of environments tested.
- We show that RAD significantly *improves test-time generalization* on several environments in the OpenAI ProcGen benchmark suite [31] widely used for generalization in RL.
- We *introduce two new data augmentations*: **random translation** for image-based input and **random amplitude scaling** for proprioceptive input that are utilized to achieve state-of-the-art results. To the best of our knowledge, these augmentations were not used in prior work.

## 4.2 Related work

### Data augmentation in computer vision

Data augmentation in deep learning systems for computer vision can be found as early as LeNet-5 [113], an early implementation of CNNs on MNIST digit classification. In AlexNet [104] wherein the authors applied CNNs to image classification on ImageNet [37], data augmentations, such as random flip and crop, were used to improve the classification accuracy. These data augmentations inject the priors of invariance to translation and reflection, playing a significant role in improving the performance of supervised computer vision systems. Recently, new augmentation techniques such as AutoAugment [33] and RandAugment [34] have been proposed to further improve the performance. For unsupervised and semi-supervised learning, several unsupervised data augmentation techniques have been proposed [16, 200, 163]. In particular, contrastive representation learning approaches [79, 26, 76] with data augmentations have recently dramatically improved the label-efficiency of downstream vision tasks like ImageNet classification.

### Data augmentation in reinforcement learning

Data augmentation has also been investigated in the context of RL though, to the best of our knowledge, there was no extensive study on a variety of widely used benchmarks prior to this work.

For improving generalization in RL, domain randomization [151, 182, 141] was proposed to transfer policies from simulation to the real world by utilizing diverse simulated experiences. Cobbe et al. [32] and Lee et al showed that simple data augmentation techniques such as cutout [32] and random convolution can be useful to improve generalization of agents on the OpenAI CoinRun and ProcGen benchmarks.

To improve the data-efficiency, CURL [164] utilized data augmentations for learning contrastive representations in the RL setting. While the focus in CURL was to make use of data augmentations jointly through contrastive and reinforcement learning losses, RAD attempts to directly use data augmentations for reinforcement learning without any auxiliary loss (see Section 4.10 for discussions on tradeoffs between CURL and RAD). Concurrent and independent to our work, DrQ [103] utilized random cropping and regularized Q-functions in conjunction with the off-policy RL algorithm SAC [67]. On the other hand, RAD can be plugged into any reinforcement learning method (on-policy methods like PPO [156] and off-policy methods like SAC [67]) *without making any changes to the underlying algorithm*.

For a more detailed and comprehensive discussion of prior work, we refer the reader to Appendix 4.7.

## 4.3 Background

RL agents act within a Markov Decision Process, defined as the tuple $(\mathcal{S}, \mathcal{A}, P, \gamma)$, with the following components: states $s \in \mathcal{S} = \mathbb{R}^n$, actions $a \in \mathcal{A}$, and state transition distribution, $P = P(s_{t+1}, r_t | s_t, a_t)$, which defines the task mechanics and rewards. Without prior knowledge of $P$, the RL agent's goal is to use experience to maximize expected rewards, $R = \sum_{t=0}^{\infty} \gamma^t r_t$, under discount factor $\gamma \in [0, 1)$. Crucially, in RL from pixels, the agent receives image-based observations, $o_t = O(s_t) \in \mathbb{R}^k$, which are a high-dimensional, indirect representation of the state.

**Soft Actor-Critic**. SAC [67] is a state-of-the-art off-policy algorithm for continuous controls. SAC learns a policy $\pi_\psi(a|o)$ and a critic $Q_\phi(o, a)$ by maximizing a weighted objective of the reward and the policy entropy, $\mathbb{E}_{s_t, a_t \sim \pi}\left[\sum_t r_t + \alpha \mathcal{H}(\pi(\cdot|o_t))\right]$. The critic parameters are learned by minimizing the squared Bellman error using transitions $\tau_t = (o_t, a_t, o_{t+1}, r_t)$ from an experience buffer $\mathcal{D}$,

$$\mathcal{L}_Q(\phi) = \mathbb{E}_{\tau \sim \mathcal{D}}\left[\left(Q_\phi(o_t, a_t) - (r_t + \gamma V(o_{t+1}))\right)^2\right]. \tag{4.1}$$

The target value of the next state can be estimated by sampling an action using the current policy:

$$V(o_{t+1}) = \mathbb{E}_{a' \sim \pi}\left[Q_{\bar{\phi}}(o_{t+1}, a') - \alpha \log \pi_\psi(a'|o_{t+1})\right], \tag{4.2}$$

where $Q_{\bar{\phi}}$ represents a more slowly updated copy of the critic. The policy is learned by minimizing the divergence from the exponential of the soft-Q function at the same states:

$$\mathcal{L}_\pi(\psi) = -\mathbb{E}_{a \sim \pi}\left[Q_\phi(o_t, a) - \alpha \log \pi_\psi(a|o_t)\right], \tag{4.3}$$

via the reparameterization trick for the newly sampled action. $\alpha$ is learned against a target entropy.

**Proximal policy optimization**. PPO [156] is a state-of-the-art on-policy algorithm for learning a continuous or discrete control policy, $\pi_\theta(a|o)$. PPO forms policy gradients using action-advantages, $A_t = A^\pi(a_t, s_t) = Q^\pi(a_t, s_t) - V^\pi(s_t)$, and minimizes a clipped-ratio loss over minibatches of recent experience (collected under $\pi_{\theta_{old}}$):

$$\mathcal{L}_\pi(\theta) = -\mathbb{E}_{\tau\sim\pi}\left[\min\left(\rho_t(\theta)A_t,\ \text{clip}(\rho_t(\theta), 1-\epsilon, 1+\epsilon)A_t\right)\right], \quad \rho_t(\theta) = \frac{\pi_\theta(a_t|o_t)}{\pi_{\theta_{old}}(a_t|o_t)}. \tag{4.4}$$

Our PPO agents learn a state-value estimator, $V_\phi(s)$, which is regressed against a target of discounted returns and used with Generalized Advantage Estimation [156]:

$$\mathcal{L}_V(\phi) = \mathbb{E}_{\tau\sim\pi}\left[\left(V_\phi(o_t) - V_t^{targ}\right)^2\right]. \tag{4.5}$$

## 4.4 Reinforcement learning with augmented data

We investigate the utility of data augmentations in model-free RL for both off-policy and on-policy settings by processing image observations with stochastic augmentations before passing them to the agent for training. For the base RL agent, we use SAC [67] and PPO [156] as the off-policy and on-policy RL methods respectively. During training, we sample observations from either a replay buffer or a recent trajectory and augment the images within the minibatch. In the RL setting, it is common to stack consecutive frames as observations to infer temporal information such as object velocities. Crucially, augmentations are applied randomly across the batch but consistently across the frame stack [164] as shown in Figure 4.1.[1] This enables the augmentation to retain temporal information present across the frame stack.

**Augmentations of image-based input:** Across our experiments, we investigate and ablate crop, translate, window, grayscale, cutout, cutout-color, flip, rotate, random convolution, and color jitter augmentations, which are shown in Figure 4.1. Of these, the *translate and window are novel augmentations* that we did not encounter in prior work.

**Crop**: Extracts a random patch from the original frame. As our experiments will confirm, the intuition behind random cropping is primarily to imbue the agent with additional translation invariance. **Translate**: *random translation* renders the full image within a larger frame and translates the image randomly across the larger frame. For example, a $100\times100$ pixel image could be randomly translated within a $108 \times 108$ empty frame. For example, in DMControl we render $100 \times 100$ pixel frames and crop randomly to $84 \times 84$ pixels. **Window**: Selects a random window from an image by masking out the cropped part of the image. **Grayscale**: Converts RGB images to grayscale with some random probability $p$. **Cutout**: Randomly inserts a small black occlusion into the frame, which may be perceived as cutting out a small patch from the originally rendered frame. **Cutout-color**: Another variant of cutout where instead of rendering black, the occlusion color is randomly generated. **Flip**: Flips an image at random across the vertical axis. **Rotate**: Randomly samples an

[1] For on-policy RL methods such as PPO, we apply the different augmentations across the batch but consistently across time.

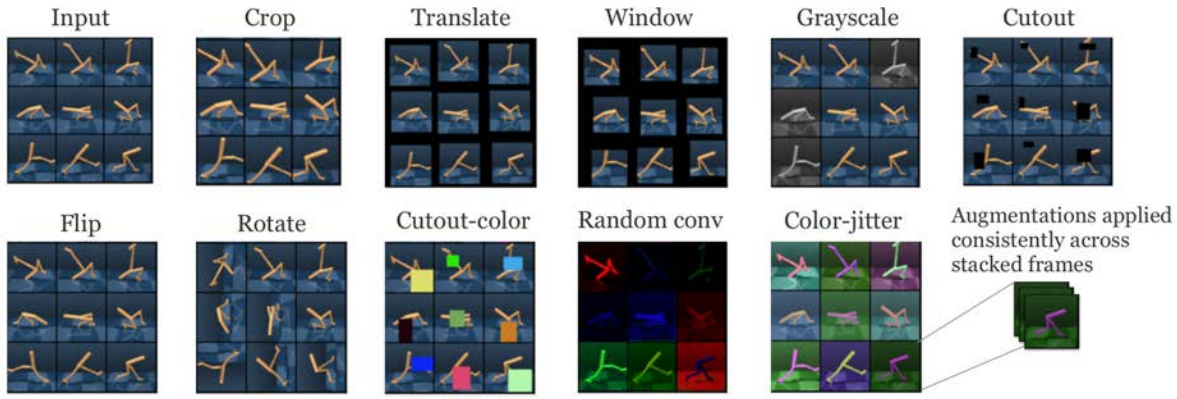



Figure 4.1: We investigate ten different types of data augmentations - crop, translate, window, grayscale, cutout, cutout-color, flip, rotate, random convolution, and color-jitter. During training, a minibatch is sampled from the replay buffer or a recent trajectory randomly augmented. While augmentation across the minibatch is stochastic, it is consistent across the stacked frames.

angle from the following set $\{0 90 180 270$ and rotates the image accordingly. **Random convolution**: Augments the image color by passing the input observation through a random convolutional layer. **Color jitter**: Converts RGB image to HSV and adds noise to the HSV channels, which results in explicit color jittering.

## 4.5 Experimental results

### Setup

**DMControl:** First, we focus on studying the data-efficiency of our proposed methods on pixel-based RL. To this end, we utilize the DeepMind Control Suite (DMControl) [177], which has recently become a common benchmark for comparing efficient RL agents, both model-based and model-free. DMControl presents a variety of complex tasks including bipedal balance, locomotion, contact forces, and goal-reaching with both sparse and dense reward signals. For DMControl experiments, we evaluate the data-efficiency by measuring the performance of our method at 100k (i.e., low sample regime) and 500k (i.e., asymptotically optimal regime) *simulator or environment* steps[2] during training by following the setup in CURL [164]. These benchmarks are referred to as *DMControl100k* and *DMControl500k*. For comparison, we consider six powerful recent pixel-based methods: CURL [164] learns contrastive representations, SLAC [114] learns a forward model and uses it to shape encoder representations, while SAC+AE [202] minimizes a reconstruction loss as

[2] *environment steps* refers to the number of times the underlying simulator is stepped through. This measure is independent of policy heuristics such as action repeat. For example, if action repeat is set to 4, then 100k *environment* steps corresponds to 25k *policy* steps.

an auxiliary task. All three methods use SAC [67] as their base algorithm. Dreamer [69] and PlaNet [70] learn world models and use them to generate synthetic rollouts similar to Dyna [170]. Pixel SAC is a vanilla Soft Actor-Critic operating on pixel inputs, and state SAC is an *oracle* baseline that operates on the proprioceptive state of the simulated agent, which includes joint positions and velocities. We also provide learning curves for longer runs and examine how RAD compares to state SAC and CURL across a more diverse set of environments in Appendix **??**.

**ProcGen:** Although DMControl is suitable for benchmarking data-efficiency and performance, it evaluates the performance on the same environment in which the agent was trained and is thus not applicable for studying generalization. For this reason, we focus on the OpenAI ProcGen benchmarks [31] to investigate the generalization capabilities of RAD. ProcGen presents a suite of game-like environments where the train and test environments differ in visual appearance and structure. For this reason, it is a commonly used benchmark for studying the generalization abilities of RL agents [32]. Specifically, we evaluate the zero-shot performance of the trained agents on the full distribution of unseen levels. Following the setup in ProcGen [31], we use the CNN architecture found in IMPALA [48] as the policy network and train the agents using the Proximal Policy Optimization (PPO) [156] algorithm for 20M timesteps. For all experiments, we use the *easy environment difficulty* and the hyperparameters suggested in [31], which have been shown to be empirically effective.

**OpenAI Gym.** For OpenAI Gym experiments with proprioceptive inputs (e.g., positions and velocities), we compare to PETS [30], a model-based RL algorithm that utilizes ensembles of dynamics models; POPLIN-P [189], a state-of-the-art model-based RL algorithm which uses a policy network to generate actions for planning; POPLIN-A [189], variant of POPLIN-P which adds noise in the action space; METRPO [105], model-based policy optimization based on TRPO [157]; and two state-of-the-art model-free algorithms, TD3 [52] and SAC [67]. In our experiments, we apply RAD to SAC. Following the experimental setups in POPLIN [189], we report the mean and standard deviation across four runs on Cheetah, Walker, Hopper, Ant, Pendulum and Cartpole environments.

Table 4.1: We report scores for RAD and baseline methods on DMControl100k and DMControl500k. In both settings, RAD achieves state-of-the-art performance on all (**6** out of **6**) environments. We selected these 6 environments for benchmarking due to availability of baseline performance data from CURL [164], PlaNet [70], Dreamer [69], SAC+AE [202], and SLAC [114]. Results are reported as averages across 10 seeds for the 6 main environments.

| 500K STEP SCORES | RAD | CURL | PLANET | DREAMER | SAC+AE | SLACV1 | PIXEL SAC | STATE SAC |
|---|---|---|---|---|---|---|---|---|
| FINGER, SPIN | **947** ± 101 | 926 ± 45 | 561 ± 284 | 796 ± 183 | 884 ± 128 | 673 ± 92 | 192 ± 166 | 923 ± 211 |
| CARTPOLE, SWING | **863** ± 9 | 845 ± 45 | 475 ± 71 | 762 ± 27 | 735 ± 63 | - | 419 ± 40 | 848 ± 15 |
| REACHER, EASY | **955** ± 71 | 929 ± 44 | 210 ± 44 | 793 ± 164 | 627 ± 58 | - | 145 ± 30 | 923 ± 24 |
| CHEETAH, RUN | **728** ± 71 | 518 ± 28 | 305 ± 131 | 570 ± 253 | 550 ± 34 | 640 ± 19 | 197 ± 15 | 795 ± 30 |
| WALKER, WALK | **918** ± 16 | 902 ± 43 | 351 ± 58 | 897 ± 49 | 847 ± 48 | 842 ± 51 | 42 ± 12 | 948 ± 54 |
| CUP, CATCH | **974** ± 12 | 959 ± 27 | 460 ± 380 | 879 ± 87 | 794 ± 58 | 852 ± 71 | 312 ± 63 | 974 ± 33 |
| 100K STEP SCORES | | | | | | | | |
| FINGER, SPIN | **856** ± 73 | 767 ± 56 | 136 ± 216 | 341 ± 70 | 740 ± 64 | 693 ± 141 | 224 ± 101 | 811 ± 46 |
| CARTPOLE, SWING | **828** ± 27 | 582 ± 146 | 297 ± 39 | 326 ± 27 | 311 ± 11 | - | 200 ± 72 | 835 ± 22 |
| REACHER, EASY | **826** ± 219 | 538 ± 233 | 20 ± 50 | 314 ± 155 | 274 ± 14 | - | 136 ± 15 | 746 ± 25 |
| CHEETAH, RUN | **447** ± 88 | 299 ± 48 | 138 ± 88 | 235 ± 137 | 267 ± 24 | 319 ± 56 | 130 ± 12 | 616 ± 18 |
| WALKER, WALK | **504** ± 191 | 403 ± 24 | 224 ± 48 | 277 ± 12 | 394 ± 22 | 361 ± 73 | 127 ± 24 | 891 ± 82 |
| CUP, CATCH | **840** ± 179 | 769 ± 43 | 0 ± 0 | 246 ± 174 | 391 ± 82 | 512 ± 110 | 97 ± 27 | 746 ± 91 |

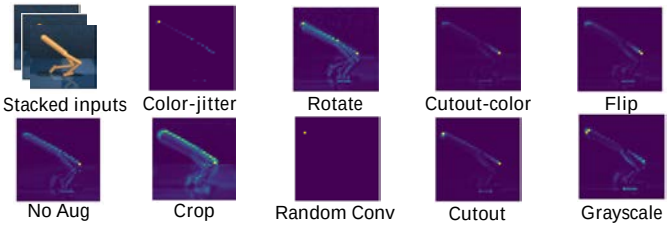
## Improving data-efficiency on DeepMind Control Suite

**Data-efficiency:** Mean scores shown in Table 4.1 and learning curves in Figure **??** show that data augmentation significantly improves the data-efficiency and performance across the six extensively benchmarked environments compared to existing methods. We summarize the main findings below:

- RAD is the **state-of-the-art algorithm** on all (**6** out of **6**) environments on both DMControl100k and DMControl500k benchmarks.
- RAD **improves** the performance of **pixel SAC by 4x** on both DMControl100k and DMControl500k *solely through data augmentation* without learning a forward model or any other auxiliary task.

| | Crop | Gray scale | Rotate | Cutout | Color-jitter | Flip |
|---|---|---|---|---|---|---|
| Crop | 920 | 849 | 635 | 855 | 797 | 650 |
| Grayscale | 856 | 175 | 349 | 187 | 214 | 231 |
| Rotate | 604 | 403 | 268 | 293 | 392 | 394 |
| Cutout | 722 | 206 | 376 | 215 | 31 | 284 |
| Color-jitter | 831 | 227 | 420 | 407 | 194 | 265 |
| Flip | 828 | 210 | 391 | 244 | 264 | 223 |

(a) Scores on DMControl500k for Walker, walk.




(b) Spatial attention map for Walker, walk.

Figure 4.2: (a) We ablate six common data augmentations on the walker, walk environment by measuring performance on DMControl500k of each permutation of any two data augmentations being performed in sequence. For example, the *crop* row and *grayscale* column correspond to the score achieved after applying random crop and then random grayscale to the input images (entries along the main axis use only one application of the augmentation). (b) Spatial attention map of an encoder that shows where the agent focuses on in order to make a decision in Walker Walk environment. Random crop enables the agent to focus on the robot body and ignore irrelevant scene details compared to other augmentations as well as the base agent that learns without any augmentation.

- RAD **matches** the performance of **state-based SAC** on the majority of (**11** out of **15**) DMControl environments tested as shown in Figure **??**.
- **Random translation** or **random crop**, stand-alone, have the highest impact on final performance relative to all other augmentations as shown in Figure **??**.

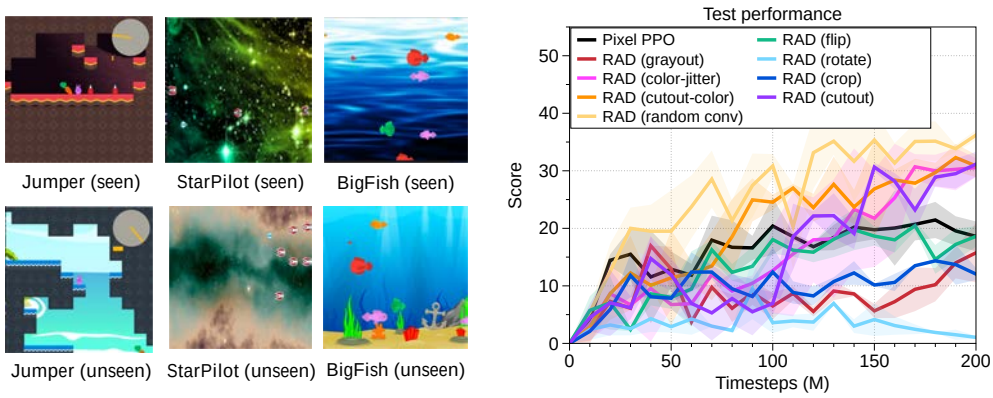
**Which data augmentations are the most effective?** To answer this question for DMControl, we ran RAD with permutations of two data augmentations applied in sequence (e.g., crop followed by grayscale) on the *Walker Walk* environment and report the scores at 500k environment steps. We chose this environment because SAC without augmentation fails at this task, resulting in scores well below 200. Our results, shown in Figure **??**, indicate that most data augmentations improve the performance of the base policy, and that random crop by itself was the most effective by a large margin.

**Why is random crop so effective?** To analyze the effects of random crop, we decompose it into its two component augmentations: (a) *random window*, which masks a random boundary region of the image, exactly where crop would remove it, and (b) *random translate*, which places the full image entirely within a larger frame of zeros, but at a random location. In Appendix **??**, Figure **??** shows resulting learning curves from each augmentation. The benefit of translations is clearly demonstrated, whereas the random information hiding due to windowing produced little effect. Table 4.1 reports scores using *random translate*, a new SOTA method, for all environments except for *Walker Walk*, where random crop sometimes reduced variance. In Figure **??** of Appendix **??**, we ablate the size of the final translation image, finding in some cases that random placement within as little as two additional pixels in height and width is sufficient to reap the benefit.

Table 4.2: We present the generalization results of RAD with different data augmentation methods on the three OpenAI ProcGen environments: BigFish, StarPilot and Jumper. We report the test performances after 20M timesteps. The results show the mean and standard deviation averaged over three runs. We see that RAD is able to outperform the baseline PPO trained on two times the number of training levels benefitting from data augmentations such as random crop, cutout and color jitter.

| | # of training levels | Pixel PPO | RAD (gray) | RAD (flip) | RAD (rotate) | RAD (random conv) | RAD (color-jitter) | RAD (cutout) | RAD (cutout-color) | RAD (crop) |
|---|---|---|---|---|---|---|---|---|---|---|
| BigFish | 100 | 1.9 ± 0.1 | 1.5 ± 0.3 | 2.3 ± 0.4 | 1.9 ± 0.0 | 1.0 ± 0.1 | 1.0 ± 0.1 | 2.9 ± 0.2 | 2.0 ± 0.2 | **5.4** ± 0.5 |
| | 200 | 4.3 ± 0.5 | 2.1 ± 0.3 | 3.5 ± 0.4 | 1.5 ± 0.6 | 1.2 ± 0.1 | 1.5 ± 0.2 | 3.3 ± 0.2 | 3.5 ± 0.3 | **6.7** ± 0.8 |
| StarPilot | 100 | 18.0 ± 0.7 | 10.6 ± 1.4 | 13.1 ± 0.2 | 9.7 ± 1.6 | 7.4 ± 0.7 | 15.0 ± 1.1 | 17.2 ± 2.0 | **22.4** ± 2.1 | 20.3 ± 0.7 |
| | 200 | 20.3 ± 0.7 | 20.6 ± 1.0 | 20.7 ± 3.9 | 15.7 ± 0.7 | 11.0 ± 1.5 | 20.6 ± 1.1 | 24.5 ± 0.1 | **24.5** ± 1.6 | 24.3 ± 0.1 |
| Jumper | 100 | 5.2 ± 0.5 | 5.2 ± 0.1 | 5.2 ± 0.7 | 5.7 ± 0.6 | 5.5 ± 0.3 | **6.1** ± 0.2 | 5.6 ± 0.1 | 5.8 ± 0.6 | 5.1 ± 0.2 |
| | 200 | **6.0** ± 0.2 | 5.6 ± 0.1 | 5.4 ± 0.3 | 5.5 ± 0.1 | 5.2 ± 0.1 | 5.9 ± 0.1 | 5.4 ± 0.1 | 5.6 ± 0.4 | 5.2 ± 0.7 |

**What are the effects on the learned encoding?** To understand how the augmentations affect learned representations encoder, we visualize a spatial attention map from the output of the last convolutional layer. Similar to [203], we compute the spatial attention map by mean-pooling the absolute values of the activations along the channel dimension and follow with a 2-dimensional spatial softmax. Figure **??** visualizes the spatial attention maps for the augmentations considered. Without augmentation, the activation is highly concentrated at the point of the forward knee, whereas with random crop/translate, entire edges of the body are prominent, providing a more complete and robust representation of the state.




(a) ProcGen (b) Test performance on modified CoinRun

Figure 4.3: (a) Examples of seen and unseen environments on ProcGen. (b) The test performance under the modified CoinRun. The solid/dashed lines and shaded regions represent the mean and standard deviation, respectively.

## Improving generalization on OpenAI ProcGen

**Generalization:** We evaluate the generalization ability on three environments from OpenAI Procgen: BigFish, StarPilot, and Jumper (see Figure **??**) by varying the number of training environments and ablating for different data augmentation methods. We summarize our findings below:

- As shown in Table 4.2, various data augmentation methods such as random crop and cutout **significantly improve the generalization performance** on the BigFish and StarPilot environments
- In particular, **RAD with random crop** achieves **55.8%** relative gain over pixel-based PPO on the BigFish environment.
- RAD trained with **100** training levels outperforms the pixel-based PPO trained with **200** training levels on both BigFish and StarPilot environments. *This shows that data augmentation can be more effective in learning generalizable representations compared to simply increasing the number of training environments.*
- In the case of Jumper (a navigation task), the gain from data augmentation is not as significant because the task involves structural generalization to different map layouts and is likely to require recurrent policies [31].
- To verify the effects of data augmentations on such environments, we consider a modified version of CoinRun [32] which corresponds to a simpler version of Jumper. We train agents on a fixed set of 500 levels with half of the available themes (style of backgrounds, floors, agents, and moving obstacles) and then measure the test performance on 1000 different levels

consisting of unseen themes to evaluate the generalization ability across the visual changes. As shown in Figure **??**, data augmentation methods, such as random convolution, color-jitter, and cutout-color, improve the generalization ability of the agent to a greater extent than random crop suggesting the need to further study data augmentations in these environments.

## Improving state-based RL on OpenAI Gym

Table 4.3: Performance on OpenAI Gym. The training timestep varies from 50,000 to 200,000 depending on the difficulty of the tasks. The results show the mean and standard deviation averaged over four runs and the best results are indicated in bold. For baseline methods, we report the best number in POPLIN [189].

| | Cheetah | Walker | Hopper | Ant | Pendulum | Cartpole |
|---|---|---|---|---|---|---|
| PETS | 2288.4 ± 1019.0 | 282.5 ± 501.6 | 114.9 ± 621.0 | 1165.5 ± 226.9 | 155.7 ± 79.3 | **199.6 ± 4.6** |
| POPLIN-A | 1562.8 ± 1136.7 | -105.0 ± 249.8 | 202.5 ± 962.5 | 1148.4 ± 438.3 | **178.3 ± 19.3** | **200.6 ± 1.3** |
| POPLIN-P | 4235.0 ± 1133.0 | 597.0 ± 478.8 | 2055.2 ± 613.8 | **2330.1 ± 320.9** | 167.9 ± 45.9 | **200.8 ± 0.3** |
| METRPO | 2283.7 ± 900.4 | -1609.3 ± 657.5 | 1272.5 ± 500.9 | 282.2 ± 18.0 | 174.8 ± 6.2 | 138.5 ± 63.2 |
| TD3 | 3015.7 ± 969.8 | -516.4 ± 812.2 | 1816.6 ± 994.8 | 870.1 ± 283.8 | 168.6 ± 12.7 | -409.2 ± 928.8 |
| SAC | 4035.7 ± 268.0 | -382.5 ± 849.5 | 2020.6 ± 692.9 | 836.5 ± 68.4 | 162.1 ± 12.3 | **199.8 ± 1.9** |
| RAD | **4554.3 ± 1209.0** | **806.4 ± 706.7** | **2149.1 ± 249.9** | 1150.9 ± 494.6 | 167.4 ± 9.7 | **199.9 ± 0.8** |
| Timesteps | 200000 | 200000 | 200000 | 200000 | 50000 | 50000 |

**Data-efficiency on state-based inputs**. Table 4.3 shows the average returns of evaluation rollouts for all methods (see Figure **??** in Appendix **??** for learning curves). We report results for state-based RAD using the best performing augmentation - random amplitude scaling; for details regarding performance of other augmentations we refer the reader to Appendix **??**. Similar to data augmentation in the visual setting, RAD is the state-of-the-art algorithm on the majority (**4** out of **6**) of benchmarked environments. RAD consistently improves the performance of SAC across all environments, and outperforms a competing state-of-the-art method - POPLIN-P - on most of the environments. It is worth noting that RAD improves the average return compared to POPLIN-P by **1.7x** in Walker, an environment where most prior RL methods fail, including both model-based and model-free ones. We hypothesize that random amplitude scaling is effective because it forces the agent to be robust to input noise while maintaining the intrinsic information of the state, such as sign of inputs and relative differences between them.

These results showcase the generality of incorporating inductive biases through augmentation (e.g., amplitude invariance) by showing that improving RL with data augmentation is not specific to pixel-based inputs but also applies RL from state. By achieving state-of-the-art performance across both visual and proprioceptive inputs, RAD sets a powerful new baseline for future algorithms.

## 4.6 Conclusion

In this work, we proposed RAD, a simple plug-and-play module to enhance any reinforcement learning (RL) method using data augmentations. For the first time, we show that data augmentations *alone* can significantly improve the data-efficiency and generalization of RL methods operating from pixels, *without any changes to the underlying RL algorithm*, on the DeepMind Control Suite and the OpenAI ProcGen benchmarks respectively. Our implementation is simple, efficient, and has been open-sourced. We hope that the performance gains, implementation ease, and wall clock efficiency of RAD make it a useful module for future research in data-efficient and generalizable RL methods; and a useful tool for facilitating real-world applications of RL.

## 4.7 Extended related work

### Data augmentation in supervised learning

Since our focus is on image-based observations, we cover the related work in computer vision. Data augmentation in deep learning systems for computer vision can be found as early as LeNet-5 [113], an early implementation of CNNs on MNIST digit classification. In AlexNet [104] wherein the authors applied CNNs to image classification on ImageNet, data augmentations were used to increase the size of the original dataset by a factor of $2048$ by randomly flipping and cropping $224 \times 224$ patches from the original image. These data augmentations inject the priors of invariance to translation and reflection, playing a significant role in improving the performance of supervised computer vision systems. Recently, new augmentation techniques such as AutoAugment [33] and RandAugment [34] have been proposed to further improve the performance of these systems.

### Data augmentation for data-efficiency in semi $\&$ self-supervised learning

Aside from improving supervised learning, data augmentation has also been widely utilized for unsupervised and semi-supervised learning. MixMatch [16], FixMatch [163], UDA [200] use unsupervised data augmentation in order to maximize label agreement without access to the actual labels. Several contrastive representation learning approaches [79, 76, 26] have recently dramatically improved the label-efficiency of downstream vision tasks like ImageNet classification. Contrastive approaches utilize data augmentations and perform patch-wise [79] or instance discrimination (MoCo, SimCLR) [76, 26]. In the instance discrimination setting, the contrastive objective aims to maximize agreement between augmentations of the same image and minimize it between all other images in the dataset [26, 76]. The choice of augmentations has a significant effect on the quality of the learned representations as demonstrated in SimCLR [26].

## Prior work in reinforcement learning related to data augmentation

### Data augmentation with domain knowledge

While not directly known for data augmentation in reinforcement learning, the following ideas can be viewed as techniques to diversify the data used to train an RL agent:

**Domain randomization** [151, 182, 141] is a simple data augmentation technique primarily used for transferring policies from simulation to the real world where one takes advantage of the simulator's access to information about the rendering and physics and thus can train transferable policies from diverse simulated experiences.

**Hindsight experience replay** [4] applies the idea of re-labeling trajectories with terminal states as fictitious goals, improving the ability of goal-conditioned RL to learn quickly with sparse rewards. This, however, makes assumptions about the goal space matching with the state space and has had limited success with pixel-based observations.

### Synthetic rollouts using a learned world model

While usually not viewed as a data augmentation technique, the idea of generating fake or synthetic rollouts to improve the data-efficiency of RL agents has been proposed in the Dyna framework [170]. In recent times, these ideas have been used to improve the performance of systems that have explicitly learned world models of the environment and generated synthetic rollouts using them [66, 95, 69].

### Data augmentation for data-efficient reinforcement learning

Data augmentation is a key component for learning contrastive representations in the RL setting as shown in the CURL framework [164], which learns representations that improve the data-efficiency of pixel-based RL by enforcing consistencies between an image and its augmented version through instance contrastive losses. Prior to our work, CURL was the state-of-the-art model for data-efficient RL from pixel inputs. While the focus in CURL was to make use of data augmentations jointly through contrastive and reinforcement learning losses, RAD attempts to directly use data augmentations for reinforcement learning without any auxiliary loss. We refer the reader to a discussion on tradeoffs between CURL and RAD in Section 4.10. Concurrent and independent to our work, DrQ [103] uses data augmentations and weighted Q-functions in conjunction with the off-policy RL algorithm SAC [67] to achieve state-of-the-art data-efficiency results on the DeepMind Control Suite. On the other hand, RAD can be plugged into any reinforcement learning method (on-policy methods like PPO [156] and off-policy methods like SAC [67]) *without making any changes to the underlying algorithm*. We further demonstrate the benefits of data augmentation to generalization on the OpenAI ProcGen benchmarks in addition to data-efficiency on the DeepMind Control Suite.

## 4.8 Code for select augmentations

```
def random_crop(imgs, size=84):
  n, c, h, w = imgs.shape
  w1 = torch.randint(0, w - size + 1, (n,))
  h1 = torch.randint(0, h - size + 1, (n,))
  cropped = torch.empty((n, c, size, size),
    dtype=imgs.dtype, device=imgs.device)
  for i, (img, w11, h11) in enumerate(zip(imgs, w1, h1)):
    cropped[i][:] = img[:, h11:h11 + size, w11:w11 + size]
  return cropped

def random_cutout(imgs, min_cut=4, max_cut=24):
  n, c, h, w = imgs.shape
  w_cut = torch.randint(min_cut, max_cut + 1, (n,)) # random size cut
  h_cut = torch.randint(min_cut, max_cut + 1, (n,)) # rectangular shape
  fills = torch.randint(0, 255, (n, c, 1, 1)) # assume uint8.
  for img, wc, hc, fill in zip(imgs, w_cut, h_cut, fills):
    w1 = torch.randint(w - wc + 1, ()) # uniform over interior
    h1 = torch.randint(h - hc + 1, ())
    img[:, h1:h1 + hc, w1:w1 + wc] = fill
  return imgs

def random_flip(imgs, p=0.5):
  n, _, _, _ = imgs.shape
  flip_mask = torch.rand(n, device=imgs.device) < p
  imgs[flip_mask] = imgs[flip_mask].flip([3]) # left-right
  return imgs
```

## 4.9 Time-efficiency of data augmentation

The primary gain of our data augmentation modules is enabling efficient augmentation of stacked frame inputs in the minibatch setting. Since the augmentations must be applied randomly across the batch but consistently across the frame stack, traditional frameworks like Tensorflow and PyTorch that focus on augmenting single-frame static datasets, are unsuitable for this task. We further show wall-clock efficiency relative to the PyTorch API in Table 4.4.

Table 4.4: We compare the data augmentation speed between the RAD augmentation modules and performing the same augmentations in PyTorch. We calculate the number of additional minutes required to perform 100k training steps. On average, the RAD augmentations are nearly 2x faster than augmentations accessed through the native PyTorch API. Additionally, since the PyTorch API is meant for processing single-frame images, it is not designed to apply augmentations consistently across the frame stack but randomly across the batch. Cutout and random convolution augmentations are not present in the PyTorch API.

| | Ours | PyTorch |
|---|---|---|
| Crop | 31.8 | 33.5 |
| Grayscale | 15.6 | 51.2 |
| Cutout | 36.6 | - |
| Cutout color | 45.2 | - |
| Flip | 4.9 | 37.0 |
| Rotate | 46.5 | 62.4 |
| Random Conv. | 45.8 | - |

## 4.10 Discussion

### CURL vs RAD

Both CURL and RAD improve the data-efficiency of RL agents by enforcing consistencies in the input observations presented to the agent. CURL does this with an explicit instance contrastive loss between an image and its augmented version using the MoCo [76] mechanism. On the other hand, RAD does not employ any auxiliary loss and directly trains the RL objective on multiple augmented views of the observations, thereby ensuring consistencies on the augmented views implicitly. The performance of RAD matches that of CURL and surpasses CURL on some of the environments in the DeepMind Control Suite (refer to Figure **??**). This suggests the potential conclusion that data augmentation is sufficient for data-efficient reinforcement learning from pixels. We argue that the conclusion requires a bit more nuance in the following subsection.

### Is data augmentation sufficient for RL from pixels?

The improved performance of RAD over CURL can be attributed to the following line of thought: While both methods try to improve the data-efficiency through augmentation consistencies (CURL explicitly, RAD implicitly); RAD outperforms CURL because *it only optimizes for what we care about, which is the task reward*. CURL, on the other hand, jointly optimizes the reinforcement and contrastive learning objectives. If the metric used to evaluate and compare these methods is the score attained on the task at hand, a method that purely focuses on reward optimization is expected

to be better as long as it implicitly ensures similarity consistencies on the augmented views (in this case, just by training the RL objective on different augmentations directly).

However, we believe that a representation learning method like CURL is *arguably* a more general framework for the usage of data augmentations in reinforcement learning. CURL can be applied *even without any task (or environment) reward* available. The contrastive learning objective in CURL that ensures consistencies between augmented views is disentangled from the reward optimization (RL) objective and is therefore capable of learning-rich semantic representations from high dimensional observations gathered from random rollouts. Real-world applications of RL might involve performing plenty of interactions (or rollouts) with sparse reward signals, and tasks presented to the agent as image-based goals. In such scenarios, CURL and other representation learning methods are *likely* to be more important even though current RL benchmarks are primarily about single or multi-task reward optimization.

Given these subtle considerations, we believe that both RAD and representation learning methods like CURL will be useful tools for an RL practitioner in future research encompassing data-efficient and generalizable RL.

## 4.11 Acknowledgments

This work was supported in part by Berkeley Deep Drive (BDD), ONR PECASE N000141612723, DARPA through the LwLL program, and the Open Philanthropy Foundation.

# Chapter 5

# Bottleneck Transformers for Visual Recognition

## 5.1 Introduction

Deep convolutional backbone architectures [104, 162, 73, 201, 175] have enabled significant progress in image classification [150], object detection [49, 121, 60, 58, 147], instance segmentation [71, 35, 75]. Most landmark backbone architectures [104, 162, 73] use multiple layers of $3 \times 3$ convolutions.

While the convolution operation can effectively capture local information, vision tasks such as object detection, instance segmentation, keypoint detection require modeling long range dependencies. For example, in instance segmentation, being able to collect and associate scene information from a large neighborhood can be useful in learning relationships across objects [88]. In order to globally aggregate the locally captured filter responses, convolution based architectures require stacking multiple layers [162, 73]. Although stacking more layers indeed improves the performance of these backbones [206], an explicit mechanism to model global (non-local) dependencies could be a more powerful and scalable solution without requiring as many layers.

Modeling long-range dependencies is critical to natural language processing (NLP) tasks as well. Self-attention is a computational primitive [186] that implements pairwise entity interactions with a content-based addressing mechanism, thereby learning a rich hierarchy of associative features across long sequences. This has now become a standard tool in the form of Transformer [186] blocks in NLP with prominent examples being GPT [142, 18] and BERT [38, 123] models.

A simple approach to using self-attention in vision is to replace spatial convolutional layers with the multi-head self-attention (MHSA) layer proposed in the Transformer [186] (Figure 5.1). This approach has seen progress on two seemingly different approaches in the recent past. On the one hand, we have models such as SASA [145], AACN [14], SANet [208], Axial-SASA [188], etc that propose to replace spatial convolutions in ResNet botleneck blocks [73] with different forms of self-attention (local, global, vector, axial, etc). On the other hand, we have the Vision Transformer (ViT) [43], that proposes to stack Transformer blocks [186] operating on linear projections of

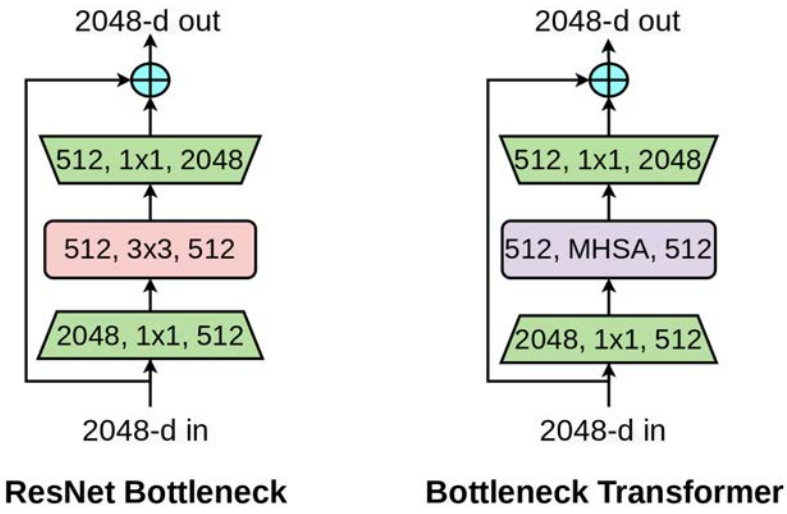



Figure 5.1: **Left**: A ResNet Bottleneck Block, **Right:** A Bottleneck Transformer (BoT) block. The only difference is the replacement of the spatial $3 \times 3$ convolution layer with Multi-Head Self-Attention (MHSA). The structure of the self-attention layer is described in Figure 5.4.

non-overlapping patches. It may appear that these approaches present two different classes of architectures. We point out that it is *not the case*. Rather, ResNet botteneck blocks with the MHSA layer can be viewed as Transformer blocks with a bottleneck structure, modulo minor differences such as the residual connections, choice of normalization layers, etc. (Figure 5.3). Given this equivalence, we call ResNet bottleneck blocks with the MHSA layer as *Bottleneck Transformer* (BoT) blocks.

Here are a few challenges when using self-attention in vision: (1) Image sizes are much larger ($1024 \times 1024$) in object detection and instance segmentation compared to image classification ($224 \times 224$). (2) The memory and computation for self-attention scale quadratically with spatial dimensions [178], causing overheads for training and inference.

To overcome these challenges, we consider the following design: (1) Use convolutions to *efficiently* learn *abstract* and *low resolution* featuremaps from large images; (2) Use global (*all2all*) self-attention to process and aggregate the information contained in the featuremaps captured by convolutions. Such a hybrid design [14] (1) uses existing and well optimized primitives for both convolutions and all2all self-attention; (2) can deal with large images efficiently by having convolutions do the spatial downsampling and letting attention work on smaller resolutions. Here is a simple practical instantiation of this hybrid design: Replace *only* the final three bottleneck blocks of a ResNet with BoT blocks *without any other changes*. Or in other words, take a ResNet and only replace the final three $3 \times 3$ convolutions with MHSA layers (Fig 5.1, Table 5.1). This simple change improves the mask AP by 1.2% on the COCO instance segmentation benchmark [121] over our canonical baseline that uses ResNet-50 in the Mask R-CNN framework [75] with *no hyperparameter differences* and minimal overheads for training and inference. Moving forward, we call this simple instantiation as BoTNet given its connections to the Transformer through the

BoT blocks. While we note that there is no novelty in its construction, we believe the simplicity and performance make it a useful reference backbone architecture that is worth studying.

Using BoTNet, we demonstrate significantly improved results on instance segmentation *without any bells and whistles* such as Cascade R-CNN [20], FPN changes [122, 54, 124, 176], hyperparameter changes [175], etc. A few key results from BoTNet are: (1) Performance gains across various training configurations (Section 5.4), data augmentations (Section 5.4) and ResNet family backbones (Section 5.4); (2) Significant boost from BoTNet on small objects (+2.4 Mask AP and +2.6 Box AP) (Appendix); (3) Performance gains over Non-Local layers (Section 5.4); (4) Gains that scale well with larger images resulting in **44.4%** mask AP, competitive with state-of-the-art performance among entries that only study backbone architectures with modest training schedules (up to 72 epochs) and no extra data or augmentations.[1].

Lastly, we scale BoTNets, taking inspiration from the training and scaling strategies in [175, 145, 116, 149, 143, 206, 15], after noting that BoTNets do not provide substantial gains in a smaller scale training regime. We design a family of BoTNet models that achieve up to **84.7%** top-1 accuracy on the ImageNet validation set, while being upto **1.64x** faster than the popular EfficientNet models in terms of *compute* time on TPU-v3 hardware. By providing *strong results* through BoTNet, we hope that self-attention becomes a widely used primitive in future vision architectures.

## 5.2 Related Work

A taxonomy of deep learning architectures that employ self-attention for vision is presented in Figure 5.2. In this section, we focus on: (1) Transformer vs BoTNet; (2) DETR vs BoTNet; (3) Non-Local vs BoTNet.

**Connection to the Transformer:** As the title of the paper suggests, one key message in this paper is that ResNet bottleneck blocks with Multi-Head Self-Attention (MHSA) layers can be viewed as Transformer blocks with a bottleneck structure. This is visually explained in Figure 5.3 and we name this block as Bottleneck Transformer (BoT). We note that the architectural design of the BoT block is not our contribution. Rather, we point out the relationship between MHSA ResNet bottleneck blocks and the Transformer with the hope that it improves our understanding of architecture design spaces [144, 143] for self-attention in computer vision. There are still a few differences aside from the ones already visible in the figure (residual connections and block boundaries): (1) Normalization: Transformers use Layer Normalization [7] while BoT blocks use Batch Normalization [90] as is typical in ResNet bottleneck blocks [73]; (2) Non-Linearities: Transformers use one non-linearity in the FFN block, while the ResNet structure allows BoT block to use three non-linearities; (3) Output projections: The MHSA block in a Transformer contains an output projection while the MHSA layer (Fig 5.4) in a BoT block (Fig 5.1) does not; (4) We use the SGD with momentum optimizer typically used in computer vision [73, 75, 59] while Transformers are generally trained with the Adam optimizer [99, 186, 23, 43].

**Connection to DETR:** Detection Transformer (DETR) is a detection framework that uses a Transformer to implicitly perform region proposals and localization of objects instead of using an

[1] SoTA is based on `https://paperswithcode.com/sota/instance-segmentation-on-coco-minival`.

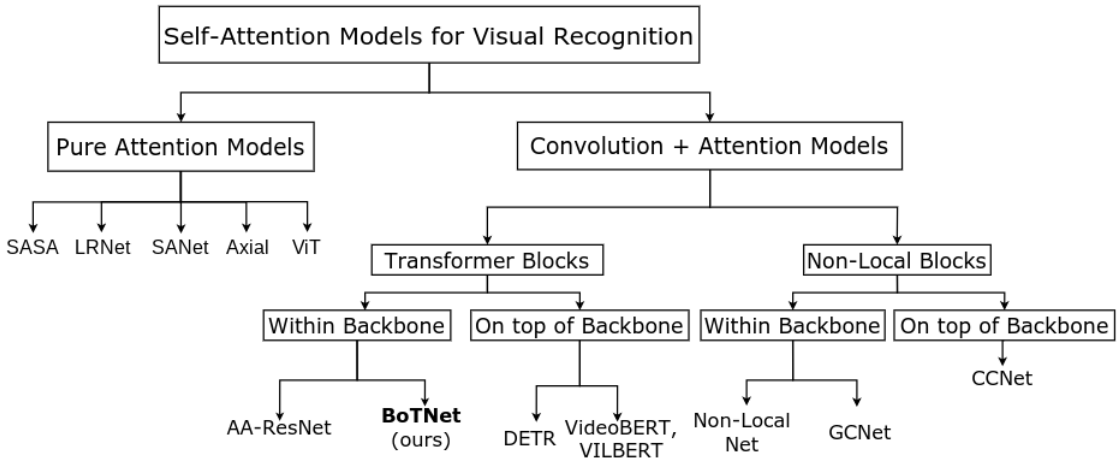



Figure 5.2: A taxonomy of deep learning architectures using self-attention for visual recognition. Our proposed architecture BoTNet is a hybrid model that uses both convolutions and self-attention. The specific implementation of self-attention could either resemble a Transformer block [186] or a Non-Local block [191] (difference highlighted in Figure 5.4). BoTNet is different from architectures such as DETR [23], VideoBERT [168], VILBERT [125], CCNet [89], etc by employing self-attention within the backbone architecture, in contrast to using them outside the backbone architecture. Being a hybrid model, BoTNet differs from pure attention models such as SASA [145], LRNet [87], SANet [208], Axial-SASA [86, 188] and ViT [43]. AA-ResNet [14] also attempted to replace a fraction of spatial convolution channels with self-attention.

R-CNN [60, 58, 147, 75]. Both DETR and BoTNet attempt to use self-attention to improve the performance on object detection and instance (or panoptic) segmentation. The difference lies in the fact that DETR uses Transformer blocks outside the backbone architecture with the motivation to get rid of region proposals and non-maximal suppression for simplicity. On the other hand, the goal in BoTNet is to provide a backbone architecture that uses Transformer-like blocks for detection and instance segmentation. We are agnostic to the detection framework (be it DETR or R-CNN). We perform our experiments with the Mask [75] and Faster R-CNN [147] systems and leave it for future work to integrate BoTNet as the backbone in the DETR framework. With visibly good gains on small objects in BoTNet, we believe there maybe an opportunity to address the lack of gain on small objects found in DETR, in future (refer to Appendix).

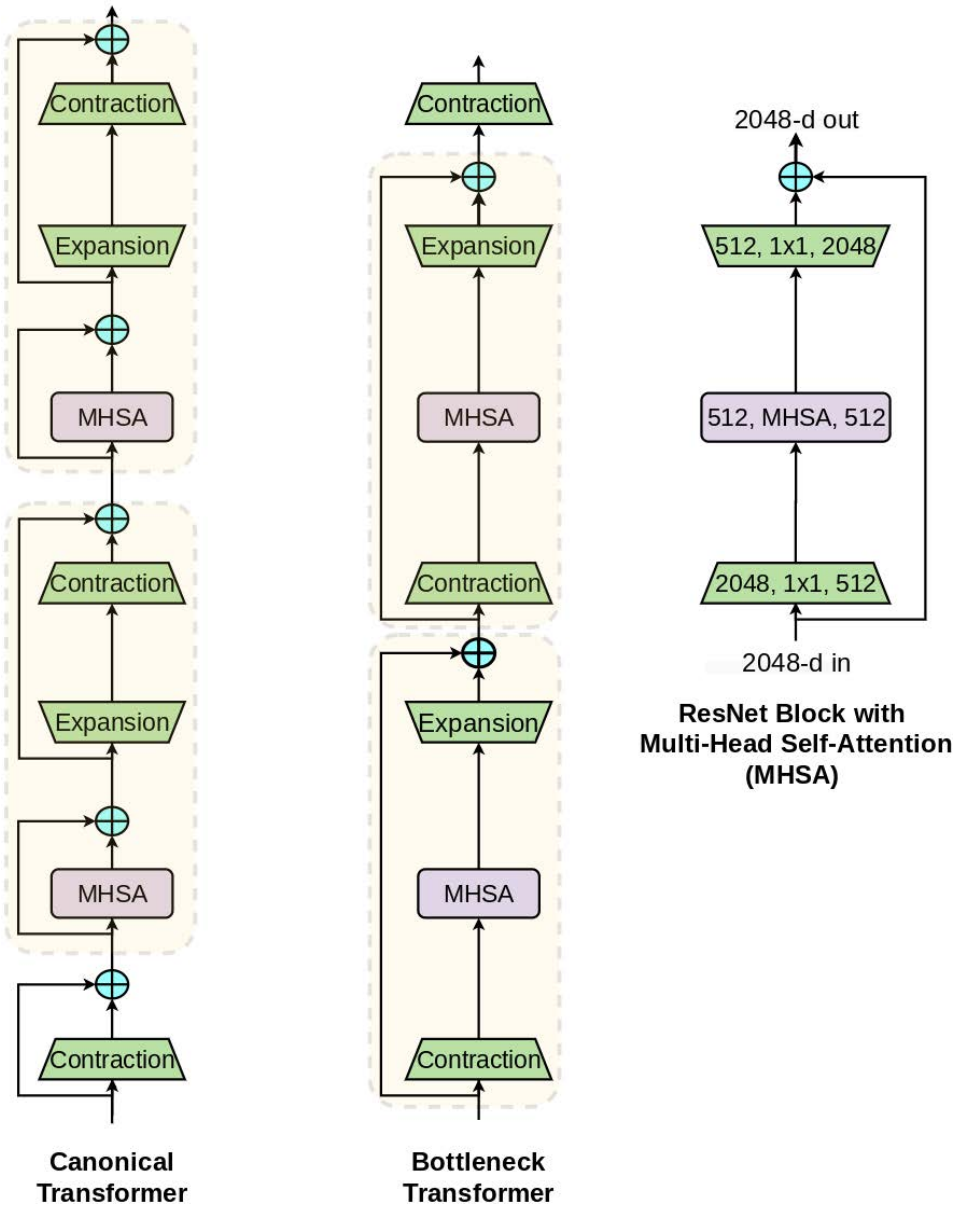



Figure 5.3: **Left:** Canonical view of the Transformer with the boundaries depicting the definition of a Transformer block as described in Vaswani et. al [186]. **Middle:** Bottleneck view of the Transformer with boundaries depicting what we define as the Bottleneck Transformer (BoT) block in this work. The architectural structure that already exists in the Transformer can be interpreted a ResNet bottleneck block [73] with Multi-Head Self-Attention (MHSA) [186] with a different notion of block boundary as illustrated. **Right:** An instantiation of the Bottleneck Transformer as a ResNet bottleneck block [73] with the difference from a canonical ResNet block being the replacement of $3 \times 3$ convolution with MHSA.

**Connection to Non-Local Neural Nets:**[2] Non-Local (NL) Nets [191] make a connection between the Transformer and the Non-Local-Means algorithm [19]. They insert NL blocks into the final one (or) two blockgroups (`c4,c5`) in a ResNet and improve the performance on video recognition and instance segmentation. Like NL-Nets [191, 21], BoTNet is a hybrid design using convolutions and global self-attention. (1) Three differences between a NL layer and a MHSA layer (illustrated in Figure 5.4): use of multiple heads, value projection and position encodings in MHSA; (2) NL blocks use a bottleneck with channel factor reduction of $2$ (instead of $4$ in BoT blocks which adopt the ResNet structure); (3) NL blocks are inserted as additional blocks into a ResNet backbone as opposed to replacing existing convolutional blocks as done by BoTNet. Section 5.4 offers a comparison between BoTNet, NLNet as well as a NL-like version of BoTNet where we insert BoT blocks in the same manner as NL blocks instead of replacing.

## 5.3 Method

BoTNet by design is simple: replace the final three spatial ($3 \times 3$) convolutions in a ResNet with Multi-Head Self-Attention (MHSA) layers that implement global (*all2all*) self-attention over a 2D featuremap (Fig 5.4). A ResNet typically has 4 stages (or blockgroups) commonly referred to as `[c2,c3,c4,c5]` with strides `[4,8,16,32]` relative to the input image, respectively. Stacks `[c2,c3,c4,c5]` consist of multiple *bottleneck* blocks with residual connections (e.g, R50 has `[3,4,6,3]` bottleneck blocks).

Approaches that use self-attention throughout the backbone [145, 14, 208, 43] are feasible for input resolutions ($224 \times 224$ (for classification) and $640 \times 640$ (for detection experiments in SASA [145])) considered in these papers. Our goal is to use attention in more realistic settings of high performance instance segmentation models, where typically images of larger resolution ($1024 \times 1024$) are used. Considering that self-attention when performed globally across $n$ entities requires $O(n^2d)$ memory and computation [186], we believe that the simplest setting that adheres to the above factors would be to incorporate self-attention at the lowest resolution featuremaps in the backbone, ie, the residual blocks in the `c5` stack. The `c5` stack in a ResNet backbone typically uses 3 blocks with one spatial $3 \times 3$ convolution in each. Replacing them with MHSA layers forms the basis of the BoTNet architecture. The first block in `c5` uses a $3 \times 3$ convolution of stride $2$ while the other two use a stride of $1$. Since `all2all` attention is not a strided operation, we use a $2 \times 2$ average-pooling with a stride $2$ for the first BoT block. The BoTNet architecture is described in Table 5.1 and the MHSA layer is presented in Figure 5.4. The strided version of the BoT block is presented in the Appendix.

**Relative Position Encodings:** In order to make the attention operation *position aware*, Transformer based architectures typically make use of a position encoding [186]. It has been observed lately that *relative-distance-aware* position encodings [159] are better suited for vision tasks [14, 145, 208]. This can be attributed to attention not only taking into account the content information

[2]The replacement vs insertion contrast has previously been pointed out in AA-ResNet (Bello et. al) [14]. The difference in our work is the complete replacement as opposed to fractional replacement in Bello et al.

| stage | output | ResNet-50 | **BoTNet-50** |
|---|---|---|---|
| c1 | 512 × 512 | 7×7, 64, stride 2 | 7×7, 64, stride 2 |
| c2 | 256 × 256 | 3×3 max pool, stride 2 | 3×3 max pool, stride 2 |
| | | $\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times3$ | $\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times3$ |
| c3 | 128 × 128 | $\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times4$ | $\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times4$ |
| c4 | 64 × 64 | $\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times6$ | $\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times6$ |
| c5 | 32 × 32 | $\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times3$ | $\begin{bmatrix} 1\times1, 512 \\ \text{MHSA}, 512 \\ 1\times1, 2048 \end{bmatrix} \times3$ |
| # params. | | **25.5**$\times10^6$ | **20.8**$\times10^6$ |
| M.Adds | | **85.4**$\times10^9$ | **102.98**$\times10^9$ |
| TPU steptime | | **786.5** ms | **1032.66** ms |

Table 5.1: Architecture of BoTNet-50 (BoT50): The only difference in BoT50 from ResNet-50 (R50) is the use of MHSA layer (Figure 5.4) in c5. For an input resolution of $1024 \times 1024$, the MHSA layer in the first block of c5 operates on $64 \times 64$ while the remaining two operate on $32 \times 32$. We also report the parameters, multiply-adds (m. adds) and training time throughput (TPU-v3 steptime on a v3-8 Cloud-TPU). BoT50 has only 1.2x more m.adds. than R50. The overhead in training throughout is 1.3x. BoT50 also has 1.2x *fewer* parameters than R50. While it may appear that it is simply the aspect of performing slightly more computations that might help BoT50 over the baseline, we show that it is not the case in Section 5.4.

but also relative distances between features at different locations, thereby, being able to effectively associate information across objects with positional awareness. In BoTNet, we adopt the 2D relative position self-attention implementation from [145, 14].

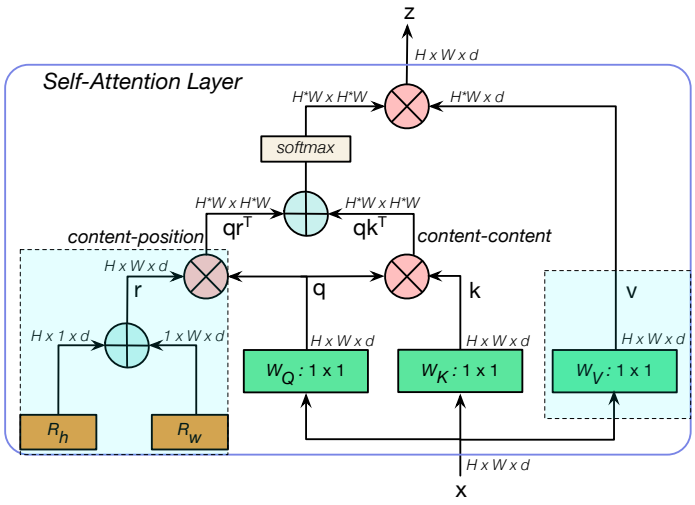



Figure 5.4: Multi-Head Self-Attention (MHSA) layer used in the BoT block. While we use 4 heads, we do not show them on the figure for simplicity. `all2all` attention is performed on a 2D featuremap with *split* relative position encodings $R_h$ and $R_w$ for height and width respectively. The attention logits are $qk^T + qr^T$ where $q, k, r$ represent query, key and position encodings respectively (we use relative distance encodings [159, 14, 145]). $\bigoplus$ and $\bigotimes$ represent element wise sum and matrix multiplication respectively, while $1 \times 1$ represents a pointwise convolution.

## 5.4 Experiments

We study the benefits of BoTNet for instance segmentation and object detection. We perform a thorough ablation study of various design choices through experiments on the COCO dataset [121]. We report the standard COCO metrics including the $AP^{bb}$ (averaged over IoU thresholds), $AP^{bb}_{50}$, $AP^{bb}_{75}$, $AP^{mk}$; $AP^{mk}_{50}$, $AP^{mk}_{75}$ for box and mask respectively. As is common practice these days, we train using the COCO `train` set and report results on the COCO `val` (or `minival`) set as followed in Detectron [59][3]. Our experiments are based on the Google Cloud TPU detection codebase[4]. We run all the baselines and ablations with the same codebase. Unless explicitly specified, our training infrastructure uses `v3-8` Cloud-TPU which contains 8 cores with 16 GB memory per core. We train with the `bfloat16` precision and cross-replica batch normalization [90, 195, 75, 59, 138] using a batch size of 64.

| Backbone | epochs | $AP^{bb}$ | $AP^{mk}$ |
|---|---|---|---|
| R50 | 12 | 39.0 | 35.0 |
| BoT50 | 12 | 39.4 (+ **0.4**) | 35.3 (+ **0.3**) |
| R50 | 24 | 41.2 | 36.9 |
| BoT50 | 24 | 42.8 (+ **1.6**) | 38.0 (+ **1.1**) |
| R50 | 36 | 42.1 | 37.7 |
| BoT50 | 36 | 43.6 (+ **1.5**) | 38.9 (+ **1.2**) |
| R50 | 72 | 42.8 | 37.9 |
| BoT50 | 72 | 43.7 (+ **0.9**) | 38.7 (+ **0.8**) |

Table 5.2: Comparing R50 and BoT50 under the 1x (12 epochs), 3x (36 epochs) and 6x (72 epochs) settings, trained with image resolution $1024 \times 1024$ and multi-scale jitter of $[0.8, 1.25]$.

## BoTNet improves over ResNet on COCO Instance Segmentation with Mask R-CNN

We consider the simplest and most widely used setting: ResNet-50[5] backbone with FPN[6]. We use images of resolution $1024 \times 1024$ with a multi-scale jitter of $[0.8, 1.25]$ (scaling the image dimension between $820$ and $1280$, in order to be consistent with the Detectron setting of using $800 \times 1300$). In this setting, we benchmark both the ResNet-50 (R50) and BoT ResNet-50 (BoT50) as the backbone architectures for multiple training schedules: **1x:** 12 epochs, **2x:** 24 epochs, **3x:** 36 epochs, **6x:** 72 epochs[7], all using the same hyperparameters for both the backbones across all the training schedules (Table 5.2). We clearly see that BoT50 is a significant improvement on top of R50 barring the 1x schedule (12 epochs). This suggests that BoT50 warrants longer training in order to show significant improvement over R50. We also see that the improvement from BoT50 in the 6x schedule (72 epochs) is worse than its improvement in the 3x schedule (32 epochs). This suggests that training much longer with the default scale jitter hurts. We address this by using a more aggressive scale jitter (Section 5.4).

## Scale Jitter helps BoTNet more than ResNet

In Section 5.4, we saw that training much longer (72 epochs) reduced the gains for BoT50. One way to address this is to increase the amount of multi-scale jitter which has been known to improve

[3] `train` - 118K images, `val` - 5K images

[4] `https://github.com/tensorflow/tpu/tree/master/models/official/detection`

[5] We use the ResNet backbones pre-trained on ImageNet classification as is common practice. For BoTNet, the replacement layers are **not** pre-trained but randomly initialized for simplicity; the remaining layers are initialized from a pre-trained ResNet.

[6] FPN refers to Feature Pyramid Network [120]. We use it in every experiment we report results on, and our FPN levels from 2 to 6 (`p2` to `p6`) similar to Detectron [59].

[7] 1x, 2x, 3x and 6x convention is adopted from MoCo [76].

| Backbone | `jitter` | $AP^{bb}$ | $AP^{mk}$ |
|---|---|---|---|
| R50 | $[0.8, 1.25]$ | 42.8 | 37.9 |
| BoT50 | $[0.8, 1.25]$ | 43.7 (+ **0.9**) | 38.7 (+ **0.8**) |
| R50 | $[0.5, 2.0]$ | 43.7 | 39.1 |
| BoT50 | $[0.5, 2.0]$ | 45.3 (+ **1.8**) | 40.5 (+ **1.4**) |
| R50 | $[0.1, 2.0]$ | 43.8 | 39.2 |
| BoT50 | $[0.1, 2.0]$ | 45.9 (+ **2.1**) | 40.7 (+ **1.5**) |

Table 5.3: Comparing R50 and BoT50 under three settings of multi-scale jitter, all trained with image resolution $1024 \times 1024$ for 72 epochs (6x training schedule).

the performance of detection and segmentation systems [45, 55]. Table 5.3 shows that BoT50 is significantly better than R50 ( + 2.1% on $AP^{bb}$ and + 1.7% on $AP^{mk}$) for multi-scale jitter of $[0.5, 2.0]$, while also showing significant gains ( + 2.2% on $AP^{bb}$ and + 1.6% on $AP^{mk}$) for scale jitter of $[0.1, 2.0]$, suggesting that BoTNet (self-attention) benefits more from extra augmentations such as multi-scale jitter compared to ResNet (pure convolutions).

## Relative Position Encodings Boost Performance

BoTNet uses relative position encodings [159]. We present an ablation for the use of relative position encodings by benchmarking the individual gains from content-content interaction ($qk^T$) and content-position interaction ($qr^T$) where $q, k, r$ represent the query, key and relative position encodings respectively. The ablations (Table 5.4) are performed with the canonical setting[8]. We see that the gains from $qr^T$ and $qk^T$ are complementary with $qr^T$ more important, ie, $qk^T$ standalone contributes to 0.6% $AP^{bb}$ and 0.6% $AP^{mk}$ improvement over the R50 baseline, while $qr^T$ standalone contributes to 1.0% $AP^{bb}$ and 0.7 % $AP^{mk}$ improvement. When combined together ($qk^T + qr^T$), the gains on both $AP^{bb}$ and $AP^{mk}$ are additive ( 1.5% and 1.2% respectively). We also see that using absolute position encodings ($qr^T_{\text{abs}}$) does not provide as much gain as relative. This suggests that introducing relative position encodings into architectures like DETR [23] is an interesting direction for future work.

[8] `res:1024x1024, 36 epochs (3x schedule), multi-scale jitter:[0.8, 1.25]`

| Backbone | Att. Type | $AP^{bb}$ | $AP^{mk}$ |
|---|---|---|---|
| R50 | - | 42.1 | 37.7 |
| BoT50 | $qk^T$ | 42.7 (+ **0.6**) | 38.3 (+ **0.6**) |
| BoT50 | $qr^T_{\text{relative}}$ | 43.1 (+ **1.0**) | 38.4 (+ **0.7**) |
| BoT50 | $qk^T + qr^T_{\text{relative}}$ | 43.6 (+ **1.5**) | 38.9 (+ **1.2**) |
| BoT50 | $qk^T + qr^T_{\text{abs}}$ | 42.5 (+ **0.4**) | 38.1 (+ **0.4**) |

Table 5.4: Ablation for Relative Position Encoding: Gains from the two types of interactions in the MHSA layers, content-content ($qk^T$) and content-position ($qr^T$).

## BoTNet improves backbones in ResNet Family

How well does the replacement setup of BoTNet work for other backbones in the ResNet family? Table 5.5 presents the results for BoTNet with R50, R101, and R152. All these experiments use the canonical training setting (refer to footnote in 5.4). These results demonstrate that BoTNet is applicable as a drop-in replacement for any ResNet backbone. Note that BoT50 is better than R101 (+ 0.3% $AP^{bb}$, **+ 0.5%** $AP^{mk}$) while it is competitive with R152 on $AP^{mk}$. Replacing 3 spatial convolutions with `all2all` attention gives more improvement in the metrics compared to stacking 50 more layers of convolutions (R101), and is competitive with stacking 100 more layers (R152), supporting our initial hypothesis that long-range dependencies are better captured through attention than stacking convolution layers.[9]

| Backbone | $AP^{bb}$ | $AP^{mk}$ |
|---|---|---|
| R50 | 42.1 | 37.7 |
| BoT50 | 43.6 (+ **1.5**) | 38.9 (+ **1.2**) |
| R101 | 43.3 | 38.4 |
| BoT101 | 45.5 (+ **2.2**) | 40.4 (+ **2.0**) |
| R152 | 44.2 | 39.1 |
| BoT152 | 46.0 (+ **1.8**) | 40.6 (+ **1.5**) |

Table 5.5: Comparing R50, R101, R152, BoT50, BoT101 and BoT152; all 6 setups using the canonical training schedule of 36 epochs, $1024 \times 1024$ images, multi-scale jitter $[0.8, 1.25]$.

## BoTNet scales well with larger images

We benchmark BoTNet as well as baseline ResNet when trained on $1280 \times 1280$ images in comparison to $1024 \times 1024$ using the best config: multi-scale jitter of $[0.1, 2.0]$ and training for 72

[9]Note that while one may argue that the improvements of BoT50 over R50 could be attributed to having 1.2x more M. Adds, BoT50 ($121 \times 10^9$ M.Adds) is also better than R101 ($162.99 \times 10^9$ B M. Adds and is competitive with R152 ($240.56 \times 10^9$ M. Adds) despite performing significantly less computation.

epochs. Results are presented in Tables 5.6 and 5.8. Results in Table 5.6 suggest that BoTNet benefits from training on larger images for all of R50, R101 and R152. BoTNet trained on $1024 \times 1024$ (leave alone $1280 \times 1280$) is significantly better than baseline ResNet trained on $1280 \times 1280$. Further, BoT200 trained with $1280 \times 1280$ achieves a $AP^{bb}$ of **49.7%** and $AP^{mk}$ of **44.4%**. We believe this result highlights the power of self-attention, in particular, because it has been achieved without any bells and whistles such as modified FPN [122, 54, 45, 176], cascade RCNN [20], etc. This result surpasses the previous best published single model single scale instance segmentation result from ResNeSt [206] evaluated on the COCO `minival` (44.2% $AP^{mk}$).

| Backbone | `res` | $AP^{bb}$ | $AP^{mk}$ |
|---|---|---|---|
| R50 | 1280 | 44.0 | 39.5 |
| BoT50 | 1024 | 45.9 (+ **1.9**) | 40.7 (+ **1.2**) |
| BoT50 | 1280 | 46.1 (+ **2.1**) | 41.2 (+ **1.8**) |
| R101 | 1280 | 46.4 | 41.2 |
| BoT101 | 1024 | 47.4 (+ **1.0**) | 42.0 (+ **0.8**) |
| BoT101 | 1280 | 47.9 (+ **1.5**) | 42.4 (+ **1.2**) |

Table 5.6: All the models are trained for 72 epochs with a multi-scale jitter of $[0.1, 2.0]$.

| Backbone | Change in backbone | $AP^{bb}$ | $AP^{mk}$ |
|---|---|---|---|
| R50 | - | 42.1 | 37.7 |
| R50 + NL [191] | + 1 NL block in `c4` | 43.1 | 38.4 |
| R50 + BoT (c4) | + 1 BoT block in `c4` | 43.7 | 38.9 |
| R50 + BoT (c4, c5) | + 2 BoT blocks in `c4,c5` | 44.9 | 39.7 |
| BoT50 | Replacement in `c5` | 43.6 | 38.9 |

Table 5.7: Comparison between BoTNet and Non-Local (NL) Nets: All models trained for 36 epochs with image size $1024 \times 1024$, jitter $[0.8, 1.25]$.

| Backbone | $AP^{bb}$ | $AP^{bb}_{50}$ | $AP^{bb}_{75}$ | $AP^{mk}$ | $AP^{mk}_{50}$ | $AP^{mk}_{75}$ |
|---|---|---|---|---|---|---|
| BoT152 | 49.5 | 71.0 | 54.2 | 43.7 | 68.2 | 47.4 |
| BoT200 | **49.7** | **71.3** | **54.6** | **44.4** | **68.9** | **48.2** |

Table 5.8: BoT152 and BoT200 trained for 72 epochs with a multi-scale jitter of $[0.1, 2.0]$.

## Comparison with Non-Local Neural Networks

How does BoTNet compare to Non-Local Neural Networks? NL ops are *inserted* into the `c4` stack of a ResNet backbone between the pre-final and final bottleneck blocks. This *adds* more parameters

to the model, whereas BoTNet ends up reducing the model parameters (Table 5.5). In the NL mould, we add ablations where we introduce BoT block in the exact same manner as the NL block. We also run an ablation with the insertion of two BoT blocks, one each in the `c4,c5` stacks. Results are presented in Table 5.7. Adding a NL improves $AP^{bb}$ by 1.0 and $AP^{bb}$ by 0.7, while adding a BoT block gives +1.6 $AP^{bb}$ and +1.2 $AP^{mk}$ showing that BoT block design is better than NL. Further, BoT-R50 (which replaces instead of adding new blocks) provides +1.5 $AP^{bb}$ and + 1.2 $AP^{mk}$, as good as adding another BoT block and better than adding one additional NL block.

## Image Classification on ImageNet

### BoTNet-S1 architecture

While we motivated the design of BoTNet for detection and segmentation, it is a natural question to ask whether the BoTNet architecture design also helps improve the image classification performance on the ImageNet [150] benchmark. Prior work [197] has shown that *adding* Non-Local blocks to ResNets and training them using canonical settings does *not* provide substantial gains. We observe a similar finding for BoTNet-50 when contrasted with ResNet-50, with both models trained with the canonical hyperparameters for ImageNet [143]: 100 epochs, batch size 1024, weight decay 1e-4, standard ResNet data augmentation, cosine learning rate schedule (Table 5.9). BoT50 does *not* provide significant gains over R50 on ImageNet though it does provide the benefit of reducing the parameters while maintaining comparable computation (M.Adds).

A simple method to fix this lack of gain is to take advantage of the image sizes typically used for image classification. In image classification, we often deal with much smaller image sizes ($224 \times 224$) compared to those used in object detection and segmentation ($1024 \times 1024$). The featuremaps on which the BoT blocks operate are hence much smaller (e.g $14 \times 14$, $7 \times 7$) compared to those in instance segmentation and detection (e.g $64 \times 64$, $32 \times 32$). With the same number of parameters, and, without a significant increase in computation, the BoTNet design in the `c5` blockgroup can be changed to uniformly use a stride of $1$ in all the final MHSA layers. We call this design as BoTNet-S1 (S1 to depict stride 1 in the final blockgroup). We note that this architecture is similar in design to the hybrid models explored in Vision Transformer (ViT) [43] that use a ResNet up to stage `c4` prior to stacking Transformer blocks. The main difference between BoTNet-S1 and the hybrid ViT models lies in the use of BoT blocks as opposed to regular Transformer blocks (other differences being normalization layer, optimizer, etc as mentioned in the contrast to Transformer in Related Work (Sec. 5.2). The architectural distinction amongst ResNet, BoTNet and BoTNet-S1, in the final blockgroup, is visually explained in the Appendix). The strided BoT block is visually explained in the Appendix.

### Evaluation in the standard training setting

We first evaluate this design for the 100 epoch setting along with R50 and BoT50. We see that BoT-S1-50 improves on top of R50 by 0.9% in the regular setting (Table 5.9). This improvement does however come at the cost of more computation (m.adds). Nevertheless, the improvement is a

promising signal for us to design models that scale well with larger images and improved training conditions that have become more commonly used since EfficientNets [175].

| Backbone | M.Adds | Params | top-1 acc. |
|---|---|---|---|
| R50 | 3.86G | 25.5M | 76.8 |
| BoT50 | 3.79G | 20.8M | 77.0 (+0.2) |
| BoT-S1-50 | 4.27G | 20.8M | 77.7 (+ **0.9**) |

Table 5.9: ImageNet results in regular training setting: 100 epochs, batch size 1024, weight decay 1e-4, standard ResNet augmentation, for all three models.

### Effect of data augmentation and longer training

We saw from our instance segmentation experiments that BoTNet and self-attention benefit more from regularization such as data augmentation (in the case of segmentation, increased multi-scale jitter) and longer training. It is natural to expect that the gains from BoT and BoT-S1 could improve when training under an improved setting: 200 epochs, batch size 4096, weight decay 8e-5, RandAugment (2 layers, magnitude 10), and label smoothing of 0.1. In line with our intuition, the gains are much more significant in this setting for both BoT50 (+ 0.6%) and BoT-S1-50 (+ 1.4%) compared to the baseline R50 (Table 5.10).

| Backbone | top-1 acc. | top-5 acc. |
|---|---|---|
| R50 | 77.7 | 93.9 |
| BoT50 | 78.3 (+ **0.6**) | 94.2 (+ **0.3**) |
| BoT-S1-50 | 79.1 (+ **1.4**) | 94.4 (+ **0.5**) |

Table 5.10: ImageNet results in an improved training setting: 200 epochs, batch size 4096, weight decay 8e-5, RandAugment (2 layers, magnitude 10), and label smoothing of 0.1

**Scaling BoTNets**

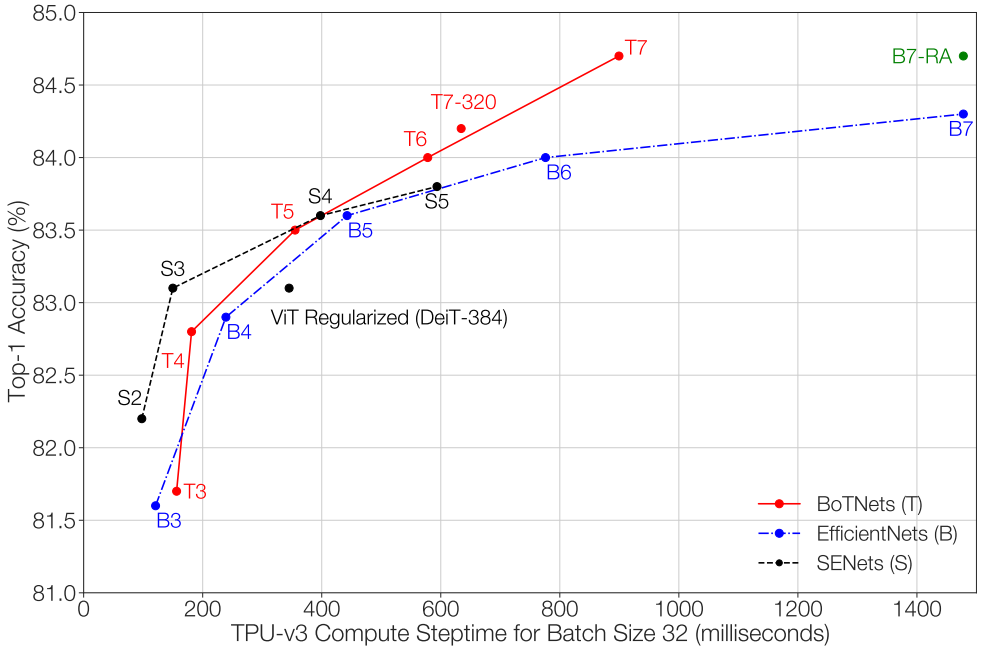



Figure 5.5: All backbones along with ViT and DeiT summarized in the form of scatter-plot and Pareto curves. SENets and BoTNets were trained while the accuracy of other models have been reported from corresponding papers.

The previous ablations show the BoNets performance with a ResNet-50 backbone and $224 \times 224$ image resolution. Here we study BoTNets when scaling up the model capacity and image resolution. There have been several works improving the performance of ConvNets on ImageNet [206, 175, 15]. Bello et al [15] recently propose scaling strategies that mainly increase model depths and increase the image resolutions much slower compared to the compound scaling rule proposed in EfficientNets [175]. We use similar scaling rules and design a family of BoTNets. The details of model depth and image resolutions are in the Appendix. We compare to the SENets baseline to understand the impact of the BoT blocks. The BoTNets and SENets experiments are performed under the same training settings (regularization and data augmentation). We additionally show EfficientNet and DeiT [183] (regularized version of ViT [43])[10] to understand the performance of BoTNets compared with popular ConvNets and Transformer models. EfficientNets and DeiT are trained under strong data augmentation, model regularization, and long training schedules, similar to the training settings of BoTNets in the experiments.

[10]ViT refers to Vision Transformer [43], while DeiT refers to Data-Efficient Image Transformer [183]. DeiT can be viewed as a regularized version of ViT with augmentations, better training hyperparameters tuned for ImageNet, and knowledge distillation [84]. We do not compare to the distilled version of DeiT since it's an orthogonal axis of improvement applicable to all models.

**ResNets and SENets are strong baselines until 83% top-1 accuracy.** ResNets and SENets achieve strong performance in the improved EfficientNet training setting. BoTNets T3 and T4 *do not* outperform SENets, while T5 does perform on par with S4. This suggests that pure convolutional models such as ResNets and SENets are still the best performing models until an accuracy regime of 83%. **BoTNets scale better beyond 83% top-1 accuracy.** While SENets are a powerful model class outperforms BoTNets (up to T4), we found gains to diminish beyond SE-350 (350 layer SENet described in Appendix) trained with image size 384. This model is referred to as S5 and achieves 83.8% top-1 accuracy. On the other hand, BoTNets scale well to larger image sizes (corroborating with our results in instance segmentation when the gains from self-attention were much more visible for larger images). In particular, T7 achieves 84.7% top-1 acc., matching the accuracy of B7-RA, with a **1.64x** speedup in efficiency. BoTNets perform better than ViT-regularized (DeiT-384), showing the power of hybrid models that make use of both convolutions and self-attention compared to pure attention models on ImageNet-1K.

## 5.5 Conclusion

The design of vision backbone architectures that use self-attention is an exciting topic. We hope that our work helps in improving the understanding of architecture design in this space. Incorporating self-attention for other computer vision tasks such as keypoint detection [22] and 3D shape prediction [61]; studying self-attention architectures for self-supervised learning in computer vision [79, 76, 180, 27, 64, 28]; and scaling to much larger datasets such as JFT, YFCC and Instagram, are ripe avenues for future research. Comparing to, and incorporating alternatives to self-attention such as lambda-layers [13] is an important future direction as well.

## 5.6 Acknowledgements

I thank Ilija Radosavovic for several useful discussions; Pengchong Jin and Xianzhi Du for help with the TF Detection codebase; Zak Stone for extensive compute support throughout this project the through TFRC program providing Google Cloud TPUs (`https://www.tensorflow.org/tfrc`).

# Chapter 6

# Conclusion

This thesis aimed to present few promising directions to improve the representation learning pipelines for perception and control. The thesis identified two axes to do so: learning objectives and deep learning architectures. The thesis took inspiration from Yann LeCun's LeCake and progress in NLP such as GPT and BERT for learning objectives. It explored contrastive representation learning for image recognition and reinforcement learning. The presented results significantly furthered and simplified the previous state-of-the-art and improved the label-efficiency and sample-efficiency of computer vision and reinforcement learning models on standardized benchmarks such as ImageNet, DMControl, Atari. The thesis also identifies the role of data augmentations in contrastive learning. It presents a detailed investigation of its role and importance in reinforcement learning and empirical tradeoffs and suitability compared to contrastive learning. As for deep learning architectures, the thesis tries to unify the architecture design of NLP and computer vision models through a simple hybrid fusion of ResNets and Transformers, presenting (then) state-of-the-art results on standardized benchmarks including ImageNet and COCO.

These are small steps towards the grand goal of having one unified architecture and learning objective across several modalities and realizing general reusable rich representations of raw data that serve as an engine for powering progress in modern AI research and deployment. The hope is that contrastive learning and self-attention are essential ingredients to the puzzle. The methods and ablations presented in this thesis are helpful for future research in this space.

For future research directions, I believe one should question absolutely everything we usually take for granted in deep learning to make the most significant contributions. For example, plenty of people told me data augmentations do not work in reinforcement learning, that their role in computer vision is not as substantial, or that it is boring to keep engineering the augmentation pipeline when working on contrastive learning. Similarly, when working on contrastive learning, many felt self-supervised learning is doomed and never likely to work as well as supervised learning. The same goes for incorporating self-attention in vision; people initially dismissed it as likely too slow for the same performance compared to convolutions. Some elements of these beliefs have valid grounding: augmentation engineering is not principled, supervised learning and pure convolutional architectures still remain a hard baseline to beat. Nevertheless, we would not have made so much progress if we only had a tunnel vision of what works *right now* and not think about what is *likely*

*to work better in the future*.

Here are a few key lessons learned while working on the articles presented in this thesis: (1) Simple baselines, when implemented carefully, are very likely to work a lot better than a fancy proposed approach one may have in mind, so it is best to always start with the simplest baseline you could think of, and throw it on the problem, and make sure all the details are right; (2) Paying as much attention to data loaders as you would for the model or the loss function - this is precisely why data augmentations are underrated - they are a big reason why we can train models that generalize well at test-time, yet, not so elegant or fancy to work on in a research project; (3) Working on methods that can scale well with more data - either computationally as architecture or as a loss function that can scale with more (unlabeled) data; (4) Investigating the scaling laws of your proposed model or loss function, across data and compute (FLOPs and model parameters).

Specifically, self-supervised learning began to shine once people started training much larger models (300M parameters or more) before figuring out a recipe that works well for modestly sized models. Further, self-supervised learning circa 2016 concluded failure by working on backbone architectures such as AlexNet or VGG and not exploring the modern state-of-the-art variants such as deep and wide ResNets. Finally, scaling with more data has enabled several recent advances such as CLIP. Therefore, it is crucial to investigate the scaling laws across data and compute frontiers to conclude the success or failure of any proposed idea.

Another specific learning from this thesis and more prominent work in the community is that few-shot learning (or semi-supervised learning) emerges for free from a well-trained (and scaled) self-supervised model. There was plenty of research in 2016-18 on meta-learning and few-shot learning, assuming that deep neural networks are incapable of being data-efficient and, therefore, one must explicitly train them to be data-efficient (meta-learning). However, it has now been shown all across computer vision, NLP, and RL that all you need is a very good initialization that can be achieved through a large-scale pre-training with a simple self-supervision objective (GPT-x, BERT, T5, CPCv2, SimCLR, DINO, CURL, APT, SPR, etc.).

While it may seem like we have made a lot of progress already, several questions remain unanswered. Contrastive learning still has its flaws in ignoring several useful properties in a scene if the augmentations make you invariant to their presence. How to precisely capture all the bits one may care about without knowing the downstream tasks in advance is a big challenge and makes unsupervised learning so exciting yet challenging. Choosing the suitable negative samples in contrastive learning is again a challenge, especially in settings like RL, where negatives across time may look too visually similar to the anchor or the positive. Investing in Siamese-style methods that can work without negatives [64] and making them general enough to work for multiple modalities and problems is a worthy direction. Here are a few questions I believe are worth considering as future research topics: (1) How can we leverage contrastive (or Siamese-like) representations for RL and robotics; (2) How can we leverage generative (or partially generative) representations for vision; (3) How can we build powerful world models for RL; (4) How could we a build a single unified multi-scale architecture that works for any modality; (5) How could we build a general-purpose AI layer that we could plug into as an API for almost any task we perform. All these are questions that currently puzzle even the best scientists, and progress in any of them is likely to be significant for realizing general-purpose intelligent machines and robots.

# Bibliography


[1] Pulkit Agrawal, Joao Carreira, and Jitendra Malik. "Learning to see by moving". In: *ICCV*. 2015.

[2] Ilge Akkaya et al. "Solving Rubik's Cube with a Robot Hand". In: *arXiv preprint arXiv:1910.07113* (2019).

[3] Ankesh Anand et al. "Unsupervised state representation learning in atari". In: *Advances in Neural Information Processing Systems*. 2019, pp. 8766–8779.

[4] Marcin Andrychowicz et al. "Hindsight Experience Replay". In: *Advances in Neural Information Processing Systems*. 2017.

[5] Relja Arandjelovic and Andrew Zisserman. "Look, listen and learn". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 609–617.

[6] Relja Arandjelovic and Andrew Zisserman. "Objects that sound". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 435–451.

[7] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. "Layer normalization". In: *arXiv preprint arXiv:1607.06450* (2016).

[8] Lei Jimmy Ba, Ryan Kiros, and Geoffrey E. Hinton. "Layer Normalization". In: *CoRR* abs/1607.06450 (2016).

[9] Philip Bachman, R Devon Hjelm, and William Buchwalter. "Learning representations by maximizing mutual information across views". In: *arXiv preprint arXiv:1906.00910* (2019).

[10] H.B. Barlow. "Unsupervised Learning". In: *Neural Computation* 1.3 (1989), pp. 295–311. DOI: 10.1162/neco.1989.1.3.295.

[11] Marc G Bellemare, Will Dabney, and Rémi Munos. "A distributional perspective on reinforcement learning". In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 449–458.

[12] Marc G Bellemare et al. "The arcade learning environment: An evaluation platform for general agents". In: *Journal of Artificial Intelligence Research* 47 (2013), pp. 253–279.

[13] Irwan Bello. "LambdaNetworks: Modeling long-range Interactions without Attention". In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=xTJEN-ggl1b.

[14] Irwan Bello et al. “Attention augmented convolutional networks”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 3286–3295.

[15] Irwan Bello et al. *Revisiting ResNets: Improved Training and Scaling Strategies*. 2021. arXiv: 2103.07579 [cs.CV].

[16] David Berthelot et al. “MixMatch: A Holistic Approach to Semi-Supervised Learning”. In: *Advances in Neural Information Processing Systems*. 2019.

[17] Greg Brockman et al. “Openai gym”. In: *arXiv preprint arXiv:1606.01540* (2016).

[18] Tom B Brown et al. “Language models are few-shot learners”. In: *arXiv preprint arXiv:2005.14165* (2020).

[19] Antoni Buades, Bartomeu Coll, and J-M Morel. “A non-local algorithm for image denoising”. In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05)*. Vol. 2. IEEE. 2005, pp. 60–65.

[20] Zhaowei Cai and Nuno Vasconcelos. “Cascade r-cnn: Delving into high quality object detection”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 6154–6162.

[21] Yue Cao et al. “Gcnet: Non-local networks meet squeeze-excitation networks and beyond”. In: *Proceedings of the IEEE International Conference on Computer Vision Workshops*. 2019, pp. 0–0.

[22] Zhe Cao et al. “Realtime multi-person 2d pose estimation using part affinity fields”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 7291–7299.

[23] Nicolas Carion et al. “End-to-End Object Detection with Transformers”. In: *arXiv preprint arXiv:2005.12872* (2020).

[24] Mathilde Caron et al. “Deep Clustering for Unsupervised Learning of Visual Features”. In: *The European Conference on Computer Vision (ECCV)*. Sept. 2018.

[25] Mathilde Caron et al. “Leveraging Large-Scale Uncurated Data for Unsupervised Pretraining of Visual Features”. In: 2019.

[26] Ting Chen et al. *A Simple Framework for Contrastive Learning of Visual Representations*. 2020. eprint: arXiv:2002.05709.

[27] Ting Chen et al. “A simple framework for contrastive learning of visual representations”. In: *arXiv preprint arXiv:2002.05709* (2020).

[28] Xinlei Chen and Kaiming He. “Exploring Simple Siamese Representation Learning”. In: *arXiv preprint arXiv:2011.10566* (2020).

[29] Sumit Chopra, Raia Hadsell, and Yann LeCun. “Learning a similarity metric discriminatively, with application to face verification”. In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2005*. 2005, pp. 539–546.

[30] Kurtland Chua et al. “Deep reinforcement learning in a handful of trials using probabilistic dynamics models”. In: *Advances in Neural Information Processing Systems*. 2018.

[31] Karl Cobbe et al. “Leveraging Procedural Generation to Benchmark Reinforcement Learning”. In: *International conference on machine learning*. 2020.

[32] Karl Cobbe et al. “Quantifying Generalization in Reinforcement Learning”. In: *International Conference on Machine Learning*. 2019.

[33] Ekin D Cubuk et al. “Autoaugment: Learning augmentation policies from data”. In: *arXiv preprint arXiv:1805.09501* (2018).

[34] Ekin D. Cubuk et al. *RandAugment: Practical automated data augmentation with a reduced search space*. 2019. eprint: arXiv:1909.13719.

[35] Jifeng Dai, Kaiming He, and Jian Sun. “Instance-aware semantic segmentation via multi-task network cascades”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 3150–3158.

[36] Jeffrey De Fauw et al. “Clinically applicable deep learning for diagnosis and referral in retinal disease”. In: *Nature medicine* 24.9 (2018), p. 1342.

[37] Jia Deng et al. “Imagenet: A large-scale hierarchical image database”. In: *Conference on Computer Vision and Pattern Recognition*. 2009.

[38] Jacob Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *CoRR* abs/1810.04805 (2018).

[39] Carl Doersch, Abhinav Gupta, and Alexei A Efros. “Unsupervised visual representation learning by context prediction”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 1422–1430.

[40] Carl Doersch and Andrew Zisserman. “Multi-task self-supervised visual learning”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 2051–2060.

[41] Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. “Adversarial feature learning”. In: *arXiv preprint arXiv:1605.09782* (2016).

[42] Jeff Donahue and Karen Simonyan. “Large Scale Adversarial Representation Learning”. In: *arXiv preprint arXiv:1907.02544* (2019).

[43] Alexey Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2020. arXiv: 2010.11929 [cs.CV].

[44] Alexey Dosovitskiy et al. “Discriminative unsupervised feature learning with convolutional neural networks”. In: *Advances in neural information processing systems*. 2014, pp. 766–774.

[45] Xianzhi Du et al. “SpineNet: Learning Scale-Permuted Backbone for Recognition and Localization”. In: *arXiv preprint arXiv:1912.05027* (2019).

[46] Yan Duan et al. "$RL^2$: Fast Reinforcement Learning via Slow Reinforcement Learning". In: *arXiv:1611.02779* (2016).

[47] Debidatta Dwibedi et al. "Learning actionable representations from visual observations". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 1577–1584.

[48] Lasse Espeholt et al. "Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures". In: *arXiv preprint arXiv:1802.01561* (2018).

[49] Mark Everingham et al. "The pascal visual object classes (voc) challenge". In: *International journal of computer vision* 88.2 (2010), pp. 303–338.

[50] Mark Everingham et al. "The PASCAL visual object classes challenge 2007 (VOC2007) results". In: (2007).

[51] Meire Fortunato et al. "Noisy networks for exploration". In: *arXiv preprint arXiv:1706.10295* (2017).

[52] Scott Fujimoto, Herke Van Hoof, and David Meger. "Addressing function approximation error in actor-critic methods". In: *International Conference on Machine Learning*. 2018.

[53] Zoubin Ghahramani, Daniel M Wolpert, and Michael I Jordan. "Generalization to local remappings of the visuomotor coordinate transformation". In: *Journal of Neuroscience* 16.21 (1996), pp. 7085–7096.

[54] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. "Nas-fpn: Learning scalable feature pyramid architecture for object detection". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 7036–7045.

[55] Golnaz Ghiasi et al. *Simple Copy-Paste is a Strong Data Augmentation Method for Instance Segmentation*. 2020. arXiv: 2012.07177 [cs.CV].

[56] Partha Ghosh et al. *From Variational to Deterministic Autoencoders*. 2019. eprint: arXiv:1903.12436.

[57] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. "Unsupervised representation learning by predicting image rotations". In: *arXiv preprint arXiv:1803.07728* (2018).

[58] Ross Girshick. "Fast r-cnn". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1440–1448.

[59] Ross Girshick et al. *Detectron*. 2018.

[60] Ross Girshick et al. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 580–587.

[61] Georgia Gkioxari, Jitendra Malik, and Justin Johnson. "Mesh r-cnn". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 9785–9795.

[62] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[63] Yves Grandvalet and Yoshua Bengio. "Semi-supervised learning by entropy minimization". In: *Advances in neural information processing systems*. 2005, pp. 529–536.

[64] Jean-Bastien Grill et al. "Bootstrap your own latent: A new approach to self-supervised learning". In: *arXiv preprint arXiv:2006.07733* (2020).

[65] Michael Gutmann and Aapo Hyvärinen. "Noise-contrastive estimation: A new estimation principle for unnormalized statistical models". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. 2010, pp. 297–304.

[66] David Ha and Jürgen Schmidhuber. "World models". In: *arXiv preprint arXiv:1803.10122* (2018).

[67] Tuomas Haarnoja et al. "Soft actor-critic algorithms and applications". In: *arXiv preprint arXiv:1812.05905* (2018).

[68] Raia Hadsell, Sumit Chopra, and Yann LeCun. "Dimensionality reduction by learning an invariant mapping". In: *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*. Vol. 2. IEEE. 2006, pp. 1735–1742.

[69] Danijar Hafner et al. "Dream to Control: Learning Behaviors by Latent Imagination". In: *arXiv preprint arXiv:1912.01603* (2019).

[70] Danijar Hafner et al. "Learning latent dynamics for planning from pixels". In: *arXiv preprint arXiv:1811.04551* (2018).

[71] Bharath Hariharan et al. "Simultaneous detection and segmentation". In: *European Conference on Computer Vision*. Springer. 2014, pp. 297–312.

[72] Hado P van Hasselt, Matteo Hessel, and John Aslanides. "When to use parametric models in reinforcement learning?" In: *Advances in Neural Information Processing Systems*. 2019, pp. 14322–14333.

[73] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[74] Kaiming He et al. "Identity mappings in deep residual networks". In: *European conference on computer vision*. Springer. 2016, pp. 630–645.

[75] Kaiming He et al. "Mask r-cnn". In: *ICCV*. 2017.

[76] Kaiming He et al. "Momentum Contrast for Unsupervised Visual Representation Learning". In: *arXiv preprint arXiv:1911.05722* (2019).

[77] Kaiming He et al. "Momentum Contrast for Unsupervised Visual Representation Learning". In: (2019). eprint: `arXiv:1911.05722`.

[78] Olivier J Hénaff, Robbe LT Goris, and Eero P Simoncelli. "Perceptual straightening of natural videos". In: *Nature neuroscience* 22.6 (2019), pp. 984–991.

[79] Olivier J Hénaff et al. "Data-efficient image recognition with contrastive predictive coding". In: *arXiv preprint arXiv:1905.09272* (2019).

[80] Peter Henderson et al. “Deep reinforcement learning that matters”. In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[81] Matteo Hessel et al. *Rainbow: Combining Improvements in Deep Reinforcement Learning*. 2017. eprint: arXiv:1710.02298.

[82] Irina Higgins et al. “Darla: Improving zero-shot transfer in reinforcement learning”. In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 1480–1490.

[83] G.E. Hinton et al. *Unsupervised Learning: Foundations of Neural Computation*. A Bradford Book. MIT Press, 1999. ISBN: 9780262581684.

[84] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. “Distilling the knowledge in a neural network”. In: *arXiv preprint arXiv:1503.02531* (2015).

[85] R Devon Hjelm et al. “Learning deep representations by mutual information estimation and maximization”. In: *arXiv preprint arXiv:1808.06670* (2018).

[86] Jonathan Ho et al. “Axial Attention in Multidimensional Transformers”. In: *arXiv preprint arXiv:1912.12180* (2019).

[87] Han Hu et al. “Local relation networks for image recognition”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 3464–3473.

[88] Han Hu et al. “Relation networks for object detection”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 3588–3597.

[89] Zilong Huang et al. “Ccnet: Criss-cross attention for semantic segmentation”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 603–612.

[90] Sergey Ioffe and Christian Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *arXiv preprint arXiv:1502.03167* (2015).

[91] Max Jaderberg et al. “Human-level performance in 3D multiplayer games with population-based reinforcement learning”. In: *Science* 364.6443 (2019), pp. 859–865.

[92] Max Jaderberg et al. “Reinforcement learning with unsupervised auxiliary tasks”. In: *arXiv preprint arXiv:1611.05397* (2016).

[93] Dinesh Jayaraman and Kristen Grauman. “Learning image representations tied to ego-motion”. In: *ICCV*. 2015.

[94] Longlong Jing and Yingli Tian. “Self-supervised Spatiotemporal Feature Learning by Video Geometric Transformations”. In: *arXiv preprint arXiv:1811.11387* (2018).

[95] Lukasz Kaiser et al. “Model-based reinforcement learning for atari”. In: *arXiv preprint arXiv:1903.00374* (2019).

[96] Dmitry Kalashnikov et al. “Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation”. In: *arXiv preprint arXiv:1806.10293* (2018).

[97] Nal Kalchbrenner et al. “Neural machine translation in linear time”. In: *arXiv preprint arXiv:1610.10099* (2016).

[98] Kacper Kielak. *Do recent advancements in model-based deep reinforcement learning really improve data efficiency?* 2020. eprint: arXiv:2003.10181.

[99] Diederik Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[100] Diederik P Kingma and Max Welling. "Auto-encoding variational bayes". In: *arXiv preprint arXiv:1312.6114* (2013).

[101] Durk P Kingma et al. "Semi-supervised learning with deep generative models". In: *Advances in neural information processing systems*. 2014, pp. 3581–3589.

[102] Alexander Kolesnikov, Xiaohua Zhai, and Lucas Beyer. "Revisiting Self-Supervised Visual Representation Learning". In: *CoRR* abs/1901.09005 (2019). arXiv: 1901.09005. URL: http://arxiv.org/abs/1901.09005.

[103] Ilya Kostrikov, Denis Yarats, and Rob Fergus. "Image augmentation is all you need: Regularizing deep reinforcement learning from pixels". In: *arXiv preprint arXiv:2004.13649* (2020).

[104] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1097–1105.

[105] Thanard Kurutach et al. "Model-ensemble trust-region policy optimization". In: *International Conference on Learning Representations*. 2018.

[106] Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. "Human-level concept learning through probabilistic program induction". In: *Science* 350.6266 (2015), pp. 1332–1338.

[107] Brenden M Lake et al. "Building machines that learn and think like people". In: *Behavioral and brain sciences* 40 (2017).

[108] Barbara Landau, Linda B Smith, and Susan S Jones. "The importance of shape in early lexical learning". In: *Cognitive development* 3.3 (1988), pp. 299–321.

[109] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. "Colorization as a proxy task for visual understanding". In: *CVPR*. 2017, pp. 6874–6883.

[110] Michael Laskin et al. "Reinforcement Learning with Augmented Data". In: *arXiv preprint arXiv:2004.14990* (2020).

[111] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *Nature* 521.7553 (2015), pp. 436–444.

[112] Yann LeCun et al. "A tutorial on energy-based learning". In: (2006).

[113] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[114] Alex X Lee et al. "Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model". In: *arXiv preprint arXiv:1907.00953* (2019).

[115] Dong-Hyun Lee. "Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks". In: *Workshop on Challenges in Representation Learning, ICML*. Vol. 3. 2013, p. 2.

[116] Jungkyu Lee et al. "Compounding the performance improvements of assembled techniques in a convolutional neural network". In: *arXiv preprint arXiv:2001.06268* (2020).

[117] Yin Li et al. "Unsupervised learning of edges". In: *CVPR*. 2016.

[118] Timothy P Lillicrap et al. "Continuous control with deep reinforcement learning". In: *arXiv preprint arXiv:1509.02971* (2015).

[119] Sungbin Lim et al. "Fast autoaugment". In: *arXiv preprint arXiv:1905.00397* (2019).

[120] Tsung-Yi Lin et al. "Feature pyramid networks for object detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 2117–2125.

[121] Tsung-Yi Lin et al. "Microsoft coco: Common objects in context". In: *European conference on computer vision*. Springer. 2014, pp. 740–755.

[122] Shu Liu et al. "Path aggregation network for instance segmentation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 8759–8768.

[123] Yinhan Liu et al. "Roberta: A robustly optimized bert pretraining approach". In: *arXiv preprint arXiv:1907.11692* (2019).

[124] Yudong Liu et al. "Cbnet: A novel composite backbone network architecture for object detection". In: *arXiv preprint arXiv:1909.03625* (2019).

[125] Jiasen Lu et al. "Vilbert: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks". In: *Advances in Neural Information Processing Systems*. 2019, pp. 13–23.

[126] Ellen M Markman. *Categorization and naming in children: Problems of induction*. mit Press, 1989.

[127] Tomas Mikolov et al. "Distributed Representations of Words and Phrases and their Compositionality". In: *Advances in Neural Information Processing Systems 26*. Ed. by C. J. C. Burges et al. Curran Associates, Inc., 2013, pp. 3111–3119.

[128] Piotr Mirowski et al. "Learning to navigate in complex environments". In: *arXiv preprint arXiv:1611.03673* (2016).

[129] Ishan Misra and Laurens van der Maaten. "Self-Supervised Learning of Pretext-Invariant Representations". In: *arXiv preprint arXiv:1912.01991* (2019).

[130] Ishan Misra, C Lawrence Zitnick, and Martial Hebert. "Shuffle and learn: unsupervised learning using temporal order verification". In: *ECCV*. 2016.

[131] Takeru Miyato et al. "Virtual adversarial training: a regularization method for supervised and semi-supervised learning". In: *IEEE transactions on pattern analysis and machine intelligence* (2018).

[132] Andriy Mnih and Koray Kavukcuoglu. "Learning word embeddings efficiently with noise-contrastive estimation". In: *Advances in neural information processing systems*. 2013, pp. 2265–2273.

[133] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), pp. 529–533.

[134] Mehdi Noroozi and Paolo Favaro. "Unsupervised learning of visual representations by solving jigsaw puzzles". In: *European Conference on Computer Vision*. Springer. 2016, pp. 69–84.

[135] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. "Representation learning with contrastive predictive coding". In: *arXiv preprint arXiv:1807.03748* (2018).

[136] Stephanie E Palmer et al. "Predictive information in a sensory population". In: *Proceedings of the National Academy of Sciences* 112.22 (2015), pp. 6908–6913.

[137] Deepak Pathak et al. "Learning Features by Watching Objects Move". In: *arXiv preprint arXiv:1612.06370* (2016).

[138] Chao Peng et al. "Megdet: A large mini-batch object detector". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 6181–6189.

[139] Lerrel Pinto, James Davidson, and Abhinav Gupta. "Supervision via competition: Robot adversaries for learning tasks". In: *arXiv preprint arXiv:1610.01685* (2016).

[140] Lerrel Pinto and Abhinav Gupta. "Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours". In: *ICRA*. 2016.

[141] Lerrel Pinto et al. "Asymmetric actor critic for image-based robot learning". In: *arXiv preprint arXiv:1710.06542* (2017).

[142] Alec Radford et al. "Language models are unsupervised multitask learners". In: *OpenAI Blog* 1.8 (2019), p. 9.

[143] Ilija Radosavovic et al. "Designing network design spaces". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 10428–10436.

[144] Ilija Radosavovic et al. "On network design spaces for visual recognition". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 1882–1890.

[145] Prajit Ramachandran et al. "Stand-alone self-attention in vision models". In: *arXiv preprint arXiv:1906.05909* (2019).

[146] Rajesh PN Rao and Dana H Ballard. "Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects". In: *Nature neuroscience* 2.1 (1999), p. 79.

[147] Shaoqing Ren et al. "Faster r-cnn: Towards real-time object detection with region proposal networks". In: *Advances in neural information processing systems*. 2015, pp. 91–99.

[148] Stefan Richthofer and Laurenz Wiskott. "Predictable feature analysis". In: *Proceedings - 2015 IEEE 14th International Conference on Machine Learning and Applications, ICMLA 2015*. 2016. ISBN: 9781509002870. DOI: 10.1109/ICMLA.2015.158. arXiv: 1311.2503.

[149] Tal Ridnik et al. "Tresnet: High performance gpu-dedicated architecture". In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 2021, pp. 1400–1409.

[150] Olga Russakovsky et al. "Imagenet large scale visual recognition challenge". In: *International journal of computer vision* 115.3 (2015), pp. 211–252.

[151] Fereshteh Sadeghi and Sergey Levine. "Cad2rl: Real single-image flight without a single real image". In: *arXiv preprint arXiv:1611.04201* (2016).

[152] Tom Schaul et al. "Prioritized experience replay". In: *arXiv preprint arXiv:1511.05952* (2015).

[153] Juergen Schmidhuber. "Making the world differentiable: On using fully recurrent self-supervised neural networks for dynamic reinforcement learning and planning in non-stationary environments." In: *Technical Report FKI-126-90, TUM* (1990).

[154] Julian Schrittwieser et al. "Mastering atari, go, chess and shogi by planning with a learned model". In: *arXiv preprint arXiv:1911.08265* (2019).

[155] Florian Schroff, Dmitry Kalenichenko, and James Philbin. "Facenet: A unified embedding for face recognition and clustering". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 815–823.

[156] John Schulman et al. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).

[157] John Schulman et al. "Trust Region Policy Optimization." In: *ICML*. 2015, pp. 1889–1897.

[158] Pierre Sermanet et al. "Time-contrastive networks: Self-supervised learning from video". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 1134–1141.

[159] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. "Self-attention with relative position representations". In: *arXiv preprint arXiv:1803.02155* (2018).

[160] Evan Shelhamer et al. "Loss is its own reward: Self-supervision for reinforcement learning". In: *arXiv preprint arXiv:1612.07307* (2016).

[161] David Silver et al. "Mastering the game of go without human knowledge". In: *Nature* 550.7676 (2017), pp. 354–359.

[162] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[163] Kihyuk Sohn et al. "FixMatch: Simplifying Semi-Supervised Learning with Consistency and Confidence". In: *arXiv:2001.07685* (2020).

[164] Aravind Srinivas, Michael Laskin, and Pieter Abbeel. "CURL: Contrastive Unsupervised Representations for Reinforcement Learning". In: *International Conference on Machine Learning*. 2020.

[165] Aravind Srinivas, Michael Laskin, and Pieter Abbeel. "Curl: Contrastive unsupervised representations for reinforcement learning". In: *arXiv preprint arXiv:2004.04136* (2020).

[166] Aravind Srinivas et al. "Universal planning networks". In: *arXiv preprint arXiv:1804.00645* (2018).

[167] Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting". In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[168] Chen Sun et al. "Videobert: A joint model for video and language representation learning". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 7464–7473.

[169] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. "Sequence to sequence learning with neural networks". In: *Advances in neural information processing systems*. 2014, pp. 3104–3112.

[170] Richard S Sutton. "Dyna, an integrated architecture for learning, planning, and reacting". In: *ACM Sigart Bulletin* 2.4 (1991), pp. 160–163.

[171] Richard S Sutton. "Integrated architectures for learning, planning, and reacting based on approximating dynamic programming". In: *Machine learning proceedings 1990*. Elsevier, 1990, pp. 216–224.

[172] Richard S Sutton et al. *Introduction to reinforcement learning*. Vol. 135. 1998.

[173] Christian Szegedy et al. "Going Deeper with Convolutions". In: *CoRR* abs/1409.4842 (2014). URL: http://arxiv.org/abs/1409.4842.

[174] Christian Szegedy et al. "Going Deeper with Convolutions". In: *Computer Vision and Pattern Recognition (CVPR)*. 2015. URL: http://arxiv.org/abs/1409.4842.

[175] Mingxing Tan and Quoc V Le. "Efficientnet: Rethinking model scaling for convolutional neural networks". In: *arXiv preprint arXiv:1905.11946* (2019).

[176] Mingxing Tan, Ruoming Pang, and Quoc V Le. "Efficientdet: Scalable and efficient object detection". In: *arXiv preprint arXiv:1911.09070* (2019).

[177] Yuval Tassa et al. "Deepmind control suite". In: *arXiv preprint arXiv:1801.00690* (2018).

[178] Yi Tay et al. "Efficient transformers: A survey". In: *arXiv preprint arXiv:2009.06732* (2020).

[179] Bart Thomee et al. "YFCC100M: The new data in multimedia research". In: *arXiv preprint arXiv:1503.01817* (2015).

[180] Yonglong Tian, Dilip Krishnan, and Phillip Isola. "Contrastive Multiview Coding". In: *arXiv preprint arXiv:1906.05849* (2019).

[181] Naftali Tishby, Fernando C Pereira, and William Bialek. "The information bottleneck method". In: *Proceedings of the 37th Annual Allerton Conference on Communication, Control and Computing (University of Illinois, Urbana, IL), Vol 37, pp 368–377.* (1999), pp. 1–16. DOI: 10.1142/S0217751X10050494. arXiv: 0004057v1 [arXiv:physics].

[182] Josh Tobin et al. "Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World". In: *arXiv preprint arXiv:1703.06907* (2017).

[183] Hugo Touvron et al. *Training data-efficient image transformers and distillation through attention*. 2021. arXiv: 2012.12877 [cs.CV].

[184] Michael Tschannen et al. "On mutual information maximization for representation learning". In: *arXiv preprint arXiv:1907.13625* (2019).

[185] Hado Van Hasselt, Arthur Guez, and David Silver. "Deep reinforcement learning with double q-learning". In: *Thirtieth AAAI conference on artificial intelligence*. 2016.

[186] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems*. 2017, pp. 5998–6008.

[187] Pascal Vincent et al. "Extracting and composing robust features with denoising autoencoders". In: *Proceedings of the 25th international conference on Machine learning*. 2008, pp. 1096–1103.

[188] Huiyu Wang et al. "Axial-DeepLab: Stand-Alone Axial-Attention for Panoptic Segmentation". In: *arXiv preprint arXiv:2003.07853* (2020).

[189] Tingwu Wang and Jimmy Ba. "Exploring model-based planning with policy networks". In: *International Conference on Learning Representations*. 2020.

[190] Xiaolong Wang and Abhinav Gupta. "Unsupervised learning of visual representations using videos". In: *ICCV*. 2015.

[191] Xiaolong Wang et al. "Non-local neural networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 7794–7803.

[192] Ziyu Wang et al. "Dueling network architectures for deep reinforcement learning". In: *arXiv preprint arXiv:1511.06581* (2015).

[193] David Warde-Farley et al. "Unsupervised control through non-parametric discriminative rewards". In: *arXiv preprint arXiv:1811.11359* (2018).

[194] Laurenz Wiskott and Terrence J Sejnowski. "Slow feature analysis: Unsupervised learning of invariances". In: *Neural computation* 14.4 (2002), pp. 715–770.

[195] Yuxin Wu and Kaiming He. "Group normalization". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 3–19.

[196] Zhirong Wu et al. "Unsupervised feature learning via non-parametric instance discrimination". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 3733–3742.

[197] Cihang Xie et al. “Feature denoising for improving adversarial robustness”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 501–509.

[198] Qizhe Xie et al. “Self-training with Noisy Student improves ImageNet classification”. In: *Conference on Computer Vision and Pattern Recognition*. 2020.

[199] Qizhe Xie et al. “Unsupervised Data Augmentation”. In: *arXiv e-prints*, arXiv:1904.12848 (Apr. 2019), arXiv:1904.12848. arXiv: 1904.12848.

[200] Qizhe Xie et al. “Unsupervised Data Augmentation for Consistency Training”. In: *arXiv:1904.12848* (2019).

[201] Saining Xie et al. “Aggregated residual transformations for deep neural networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1492–1500.

[202] Denis Yarats et al. “Improving Sample Efficiency in Model-Free Reinforcement Learning from Images”. In: *arXiv preprint arXiv:1910.01741* (2019).

[203] Sergey Zagoruyko and Nikos Komodakis. “Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer”. In: *International Conference on Learning Representations*. 2017.

[204] Amir R Zamir et al. “Generic 3d representation via pose estimation and matching”. In: *European Conference on Computer Vision*. Springer. 2016, pp. 535–553.

[205] Xiaohua Zhai et al. “$S^4L$: Self-Supervised Semi-Supervised Learning”. In: *arXiv preprint arXiv:1905.03670* (2019).

[206] Hang Zhang et al. *ResNeSt: Split-Attention Networks*. 2020. arXiv: 2004.08955 [cs.CV].

[207] Richard Zhang, Phillip Isola, and Alexei A Efros. “Colorful image colorization”. In: *European conference on computer vision*. Springer. 2016, pp. 649–666.

[208] Hengshuang Zhao, Jiaya Jia, and Vladlen Koltun. *Exploring Self-attention for Image Recognition*. 2020. arXiv: 2004.13621 [cs.CV].

[209] Xiaojin Zhu and Zoubin Ghahramani. “Learning from Labeled and Unlabeled Data with Label Propagation”. In: *Technical Report CMU-CALD-02-107, Carnegie Mellon University*. 2002.

[210] Chengxu Zhuang, Alex Lin Zhai, and Daniel Yamins. “Local aggregation for unsupervised learning of visual embeddings”. In: *arXiv preprint arXiv:1903.12355* (2019).