Université de Montréal

Self-Supervision for Data Interpretability in Image Classification and Sample Efficiency in Reinforcement Learning

par Nitarshan Rajkumar

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté à la Faculté des arts et des sciences
en vue de l'obtention du grade de Maître ès sciences (M.Sc.)
en informatique

Juin, 2021

Université de Montréal
Faculté des arts et des sciences

Ce mémoire intitulé:

Self-Supervision for Data Interpretability in Image Classification and Sample Efficiency in Reinforcement Learning

présenté par:

Nitarshan Rajkumar

a été évalué par un jury composé des personnes suivantes:

Liam Paull,	président-rapporteur
Laurent Charlin,	directeur de recherche
Jian Tang,	membre du jury

Mémoire accepté le: .....

Résumé

L'apprentissage auto-surveillé (AAS), c'est-à-dire l'apprentissage de connaissances en exploitant la structure intrinsèque présente dans un ensemble de données non étiquettées, a beaucoup fait progresser l'apprentissage automatique dans la dernière décennie, et plus particulièrement dans les dernières deux années en vision informatique. Dans cet ouvrage, nous nous servons de l'AAS comme outil dans deux champs applicatifs: Pour interpréter efficacement les ensembles de données et les décisions prises par des modèles statistiques, et pour pré-entrainer un modèle d'apprentissage par renforcement pour grandement augmenter l'efficacité de son échantillonnage dans son contexte d'entraînement.

Le Chapitre 1 présente les connaissances de fond nécessaires à la compréhension du reste du mémoire. Il offre un aperçu de l'apprentissage automatique, de l'apprentissage profond, de l'apprentissage auto-surveillé et de l'apprentissage par renforcement (profond).

Le Chapitre 2 se détourne brièvement du sujet de l'auto-surveillance pour étudier comment le phénomène de la mémorisation se manifeste dans les réseaux de neurones profonds. Les observations que nous ferons seront alors utilisées comme pièces justificatives pour les travaux présentés dans le Chapitre 3. Ce chapitre aborde la manière dont l'auto-surveillance peut être utilisée pour découvrir efficacement les régularités structurelles présentes dans un ensemble de données d'entraînement, estimer le degré de mémorisation de celui-ci par le modèle, et l'influence d'un échantillon d'entraînement sur les résultats pour un échantillon-test. Nous passons aussi en revue de récents travaux touchant à l'importance de mémoriser la "longue traîne" d'un jeu de données.

Le Chapitre 4 fait la démonstration d'une combinaison d'objectifs de pré-entraînement AAS axés sur les caractéristiques des données en apprentissage par renforcement, de ce fait élevant l'efficacité d'échantillonnage à un niveau comparable à celui d'un humain. De plus, nous montrons que l'AAS ouvre la porte à de plus grands modèles, ce qui a été par le passé un défi à surmonter en apprentissage par renforcement profond.

Finalement, le Chapitre 5 conclut l'ouvrage avec un bref survol des contributions scientifiques et propose quelque avenues pour des recherches poussées dans le futur.

mots-clés: apprentissage automatique, apprentissage profond, apprentissage de représentations, apprentissage auto-surveillé, apprentissage par renforcement, généralisation

Summary

Self-Supervised Learning (SSL), or learning representations of data by exploiting inherent structure present in it without labels, has driven significant progress in machine learning over the past decade, and in computer vision in particular over the past two years. In this work, we explore applications of SSL towards two separate goals - first, as a tool for efficiently interpreting datasets and model decisions, and second, as a tool for pretraining in reinforcement learning (RL) to greatly advance sample efficiency in that setting.

Chapter 1 introduces background material necessary to understand the remainder of this thesis. In particular, it provides an overview of Machine Learning, Deep Learning, Self-Supervised Representation Learning, and (Deep) Reinforcement Learning.

Chapter 2 briefly detours away from this thesis' focus on self-supervision, to examine how the phenomena of memorization manifests in deep neural networks. These results are then used to partially justify work presented in Chapter 3, which examines how self-supervision can be used to efficiently uncover structural regularity in training datasets, and to estimate training memorization and the influence of training samples on test samples. Recent experimental work on understanding the importance of memorizing the long-tail of data is also revisited.

Chapter 4 demonstrates how a combination of SSL pretraining objectives designed for the structure of data in RL can greatly improve sample efficiency to nearly human-level performance. Furthermore, it is shown that SSL enables the use of larger models, which has historically been a challenge in deep RL.

Chapter 5 concludes by reviewing the contributions of this work, and discusses future directions.

Keywords: machine learning, deep learning, representation learning, self-supervised learning, reinforcement learning, generalization

Contents

List of Figures

List of Tables

List of Abbreviations

AI	Artificial Intelligence
AGI	Artificial General Intelligence
DL	Deep Learning
DQN	Deep Q-Network (Mnih et al., 2015)
DRL	Deep Reinforcement Learning
E2E	End-To-End Supervised Model
GPU	Graphics Processing Unit
IID	Independent and Identically Distributed
ML	Machine Learning
MLP	Multi-Layer Perceptron
NAG	Nesterov Accelerated Gradient (Nesterov, 1983)
NN	Neural Network
OOD	Out-of-distribution
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
SGD	Stochastic Gradient Descent
SGI	SPR, Goal-Condition RL, Inverse Modelling (Schwarzer et al., 2021b)
SPR	Self-Predictive Representations (Schwarzer et al., 2021a)
SSL	Self-Supervised Learning
SSS	Self-Supervised + Simple Classifier

Acknowledgments

The circumstances under which this work was done were. . . rather unexpected [1], and I am quite thankful to have had the support of many people during this time.

First, many thanks go to Laurent Charlin, who has provided complete freedom in pursuing my research interests, while also providing guidance and support in becoming a better researcher. I also thank Brady Neal for kindly providing my first opportunity to work on a research project. I'm fortunate to have worked with Max Schwarzer, who I have learned a lot from, on reinforcement learning, paper writing, and scientific skepticism. As well, thanks to Olexa Bilaniuk, for his intellectual curiosity, his unfailing support with compute clusters, and also his help translating the summary of this thesis into French.

I'd like to thank other coauthors I collaborated with: Alex Drouin, Karolina Dziugaite, Ethan Caballero, Dan Roy, Ioannis Mitliagkas, Michael Noukhovitch, Ankesh Anand, Devon Hjelm, Phil Bachman, Aaron Courville, and David Krueger.

Beyond those mentioned already, thanks to those who made Mila a friendly and welcoming environment; in particular: Christos Tsirigotis, Joseph Viviano, Louis-Pascal Xhonneux, Evan Racah, Dora Jambor, Irina Rish, Martin Weiss, Niki Howe, Pierre-Luc Bacon, Eeshan Dhekane, Khimya Khetarpal, and Zafarali Ahmed.

I'd also like to thank my coworkers from my time in industry who were supportive of me pursuing research in academia: Alexis Midon, Alper Kokmen, Guy Rittger, Kareem Kouddous, Fraser Kelton, Rohan Kshirsagar, and Dave Cummings. I look forward to eventually building AGI with some of you!

Obviously nothing I've done would be possible without the continued support of my parents and the importance they have always placed on my education; thanks to them for everything.

Finally, a special thanks goes to all the workers that ensured the continuation of society during these unprecedented times, while I myself was privileged enough to be able to stay at home with my family.

1. To the (hypothetical) future reader, these circumstances were the COVID-19 pandemic.

1 Introduction

Machine learning (ML) is a data-driven approach to characterizing statistical regularity in our observations of the world.[1] In recent years it has seen particular success via deep learning (DL), a collection of model architectures and optimization methods which have high expressive capacity and scalability with data and compute as well as a remarkable ability to *generalize* beyond training data instead of merely memorizing it. The flexibility of deep learning comes in large part from reducing inductive biases and instead learning sophisticated representations from data directly. A significant drawback to this approach is that a large amount of data is often needed to learn from, which can be especially prohibitive when this requires manual human annotation and labelling. A standard approach to addressing this issue has been through transfer learning, in which representations learned on one task can be transferred or used as an initialization for another related task. More recent advances in Self-Supervised Learning (SSL) go further through construction of novel training objectives that exploit inherent structure present in unlabelled data in order to learn strong representations. This paradigm allows for decoupling of representation learning and more task-specific learning, and also greatly increases learning signal in settings with limited labelled data.

In this work, we explore applications of SSL towards two separate goals - first, as a tool for efficiently interpreting datasets and model decisions, and second, as an approach for pretraining in reinforcement learning (RL) to greatly advance sample efficiency in that setting.

This thesis touches on each of the primary paradigms of machine learning: supervised learning, unsupervised learning, and reinforcement learning. Accordingly, a brief overview of these is provided in this chapter, as far as is necessary to understand the remainder of this work.

1. I think this is how a McKinsey consultant might put it.

1.1 (Supervised) Machine Learning

Four important components of any ML approach to a problem are: the data the model will learn from and be evaluated on, the loss function used to measure the quality of the learned model and to provide a learning signal, the optimization procedure used to train the model, and finally the structure of the model itself. We elaborate on the first three below, and dedicate the subsequent section to the structure of neural network and deep learning models that are the focus of this thesis.

Data In supervised learning we seek to learn a mapping from input data to targets: $f : X \to Y$. We do so by learning from example pairs (x, y) of this mapping, which form a training dataset $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$. For example, this could be pairs of images and corresponding classification labels. In general, modern ML methods perform better with more data; today's largest training datasets are on the order of billions of images (Zhai et al., 2021), and trillions of words (Gao et al., 2020). Most relevant to this work are two particular image classification datasets: ImageNet-1K (Russakovsky et al., 2015) and CIFAR-100 (Krizhevsky, 2009b). ImageNet-1K is a collection of 1.2 million high-resolution images scraped from the internet, grouped into 1000 natural object classes. This dataset suffers from some flaws – it disproportionately covers certain types of classes such as dogs (which are 120 of the classes), and many images contain multiple objects or are mislabelled, causing difficulty when learning from the single crude label provided per image (Beyer et al., 2020). CIFAR-100 is a collection of 50 thousand downsampled images (32x32) covering 100 natural object classes. While more computationally accessible, this also suffers from issues such as duplicated data and label errors (Barz and Denzler, 2020).

Objectives In order to solve a problem, we need to quantify performance on it; this is achieved through **objective**, **cost** or **loss functions** which we seek to minimize. For example, in classification tasks, we ultimately care about the **0-1 loss**, which is 0 for correct classification and 1 for incorrect.

$$01(y, \hat{y}) = \mathbb{1}(y \neq \hat{y})$$

Such a loss is not differentiable however, so provides no learning signal for an agent learning with the gradient-based optimization methods commonly used today. A very common **surrogate objective** for the 0-1 loss is the differentiable **cross-entropy loss** (Bridle, 1990a,b) in combination with a **softmax** operator that ensures outputs are a normalized probability distribution $\mathbf{q}$.

$$\text{CrossEntropy}(\mathbf{q}, y) = -\log q_y$$

The use of softmax for classification has an appealing probabilistic perspective (Bishop, 2006); however, these probabilities are in general misleading and unreliable when used as confidence estimates for a model (Gal and Ghahramani, 2016).

Beyond classification, another common setting is **regression**. A common objective in this setting is the **mean-squared error loss** (MSE)

$$\text{MeanSquaredError}(\hat{\mathbf{y}}, \mathbf{y}) = \|\hat{\mathbf{y}} - \mathbf{y}\|_2^2$$

Surprisingly, recent work has empirically shown that the MSE loss works equally well to softmax cross-entropy for classification tasks (Hui and Belkin, 2021; Kornblith et al., 2020).

Optimization Generally there are not closed-form equations that let us directly fit a model to a dataset for a given objective function, and we need to use some form of search process to find good settings of parameters. By far the most prominent approach to optimization in ML (and particularly deep learning) is **Gradient Descent** and methods derived from it. Assuming our model is parameterized by weights θ, and that the model is fully differentiable with respect to these weights, gradient descent updates these weights by taking small steps (controlled by a step-size γ) in the opposite direction of the gradient (steepest descent) of these weights with respect to some scalar loss evaluated on a training dataset:

$$\theta_{t+1} \leftarrow \theta_t - \gamma \nabla_{\theta_t} \text{Loss}(\theta_t, D_{train})$$

For neural networks, the **backpropagation** algorithm (Linnainmaa, 1970; Werbos, 1982; Rumelhart et al., 1986) [2] allows for layer-wise calculation of gradients in a highly efficient manner.

In practice, gradients are calculated and averaged over a minibatch of samples, to more efficiently leverage the parallelism of GPUs (graphics processing units) and allow for faster parameter updates — this approach is called **Stochastic Gradient Descent** (SGD). Furthermore, additional **momentum** terms are commonly used to speed up convergence of SGD, by increasing the gradient in directions of persistent reduction in loss over iterations (Polyak, 1964; Nesterov, 1983). Two common momentum-based SGD methods are Nesterov-accelerated-gradient (NAG Nesterov, 1983) and Adam (Kingma and Ba, 2014), which are now standard for optimization in deep learning. A vast number of variants on SGD have been proposed, though large-scale studies have shown limited benefit in terms of test performance to any beyond NAG or Adam (Schmidt et al., 2021; Choi et al., 2020; Nado et al., 2021)

1.2 Deep Learning

Deep Learning is at present the dominant focus of research in ML. Its origins can be traced to early work on biologically-inspired neural network (NN) methods such as perceptrons (Rosenblatt, 1958) [3] or the neocognitron (Fukushima, 1980), though much of today's work has little connection to learning in natural systems. Neural networks have undergone cycles of interest and disdain in the community of artificial intelligence (AI) researchers over the past sixty-five years, but over this time they have been the only approach to consistently scale with exponentially increased computational resources and data (Sutskever, 2018; Amodei et al., 2019; Hernandez and Brown, 2020). We provide a brief summary of the relevant material here; a far more thorough (though slightly out-of-date) overview of the field can be found in Goodfellow et al. (2016).

2. Backpropagation is simply an application of the chain rule of calculus and reverse-mode automatic-differentiation to neural networks with scalar loss functions.

3. Interestingly, Frank Rosenblatt (inventor of the perceptron machine) explicitly distanced himself from the goal of building artificial intelligence, and worked on the perceptron as a brain model to better understand natural intelligence (Rosenblatt, 1962)

1.2.1 Building Blocks

Central to deep learning is the *composition* of successive functions, such that sophisticated intermediary representations can be learned that enable better performance on the task at hand. Each function in the composition, referred to as a layer, can take on various forms to incorporate different inductive biases relevant to specific tasks.

$$f = l_L \circ l_{L-1} \circ \cdots \circ l_1$$

Linear The simplest layer is a linear layer, which is parameterized by a **weight** matrix and an optional **bias** vector; applying this layer to its input is merely a matrix multiplication and vector addition.

$$\text{Linear}(\mathbf{x}; \mathbf{W}, \mathbf{b}) = \mathbf{W}\mathbf{x} + \mathbf{b} \qquad \mathbb{R}^D \to \mathbb{R}^{D'}$$

Nonlinearities Composing linear layers alone does not provide any extra expressive power over a single linear layer, and accordingly cannot be used to model non-linear relationships in data. It is thus required to introduce nonlinear transformations between layers; the most widespread nonlinearity in use today is the **Rectified Linar Unit** (ReLU) applied element-wise:

$$\text{ReLU}(\mathbf{x})_i = \max(0, x_i) \qquad \mathbb{R}^D \to \mathbb{R}^D$$

Special nonlinearities are commonly used on the logits (the outputs of the penultimate layer); for example in multi-class classification, it is common to use a **Softmax** to turn the unnormalized logits into a normalized probability distribution:

$$\text{Softmax}(\mathbf{x})_i = \frac{e^{x_i}}{\sum_j e^{x_j}} \qquad \mathbb{R}^D \to \mathbb{R}^D$$

Convolutional Another very common layer is the **convolutional** layer which operates on data structured in grids; it has been historically dominant on vision tasks.

$$\text{Conv2D}(\mathbf{X}; \mathbf{K}_c)_{hwc} = \mathbf{X} \star \mathbf{K_c} \qquad \mathbb{R}^{HWC} \to \mathbb{R}^{H'W'C'}$$

where $\star$ is the cross-correlation operation and K_c is the c-th of C' kernels. The core inductive bias of convolutional layers is of spatial locality – that the same operation can be applied to all regions of an input. This allows for significant reductions in parameter count compared to equivalent fully-connected linear layers. Convolutional layers are frequently used in combination with **pooling** layers,

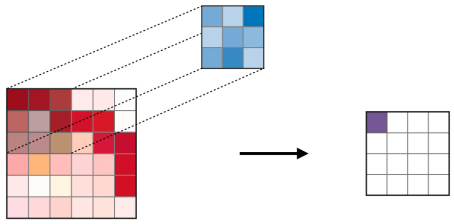


Figure 1.1 – Visualization of applying a convolutional kernel (Amidi and Amidi, 2019)

which aggregate outputs over a region with a statistic such as the maximum or average, enabling robustness to small translations in the input, as well as reducing the dimensionality of the output for downstream layers. Pooling is not strictly necessary, and networks can be fully convolutional without any pooling or linear layers (Springenberg et al., 2015).

Self-Attention A relatively recent, yet highly promising, layer is the dot-product self-attention layer (Luong et al., 2015; Vaswani et al., 2017). For an input sequence $\mathbf{z} \in \mathbb{R}^{N \times D}$ of length N, with each token having dimension D:

$$\text{SelfAttention}(\mathbf{x}; \mathbf{U_{qkv}}) = \text{Softmax}\left(\frac{(\mathbf{U_q x})(\mathbf{U_k x})^\top}{\sqrt{D_h}}\right)(\mathbf{U_v x}) \qquad \mathbb{R}^{ND} \to \mathbb{R}^{ND_h}$$

The parameters $\mathbf{U_q kv}$ project the input tokens of $\mathbf{x}$ into query $\mathbf{q}$, key $\mathbf{k}$ and value $\mathbf{v}$ representations, of the same sequence length as the input. Self-attention provides a way for information to rapidly propagate between different areas of the input; this was especially useful in the natural language processing (NLP) setting it was first introduced in (Luong et al., 2015; Vaswani et al., 2017), but self-attention has recently been used in computer vision (CV) to drive significant advances in compute efficiency and performance (Dosovitskiy et al., 2020). Unfortunately, self-attention suffers from poor computational complexity, due to the quadratic scaling of the outer product of the query $\mathbf{q}$ and the key $\mathbf{k}$; motivating recent work on approximations to self-attention that have linear computational cost (Wang et al.,

2020; Choromanski et al., 2021), and approaches for global information propagation that discard self-attention entirely and use regular linear layers and transposes or frequency domain transformations (Melas-Kyriazi, 2021; Lee-Thorp et al., 2021).

Normalization Other commonly used layers are normalization layers, which make optimization with SGD more stable for deep models. Batch Norm (Ioffe and Szegedy, 2015) is the most prominent of these layers, and operates by keeping per-channel running mean and variance statistics, calculated over the inputs of a batch, and then using these statistics to center and normalize inputs. However, there has been a move away from Batch Norm recently due to its significant drawbacks: instability with small batches, inter-device communication overhead in distributed training, and inappropriateness when transferring to new distributions (Kolesnikov et al., 2020). More recently, alternatives which normalize each input in isolation, such as Layer Norm (Ba et al., 2016), Group Norm (Wu and He, 2018), and Instance Norm (Ulyanov et al., 2017), have gained in popularity due to their ease of use and improved task and compute performance over Batch Norm. Beyond their stabilizing effect on optimization, normalization layers also have significant expressive capacity by themselves; it has been shown that good models can be learnt by training batch norm alone with the rest of the model frozen at a random initialization (Frankle et al., 2021), and that transferring a model to new modalities (e.g. language to images) can be done by finetuning input and output layers along with intermediary layer norm layers alone (Lu et al., 2021).

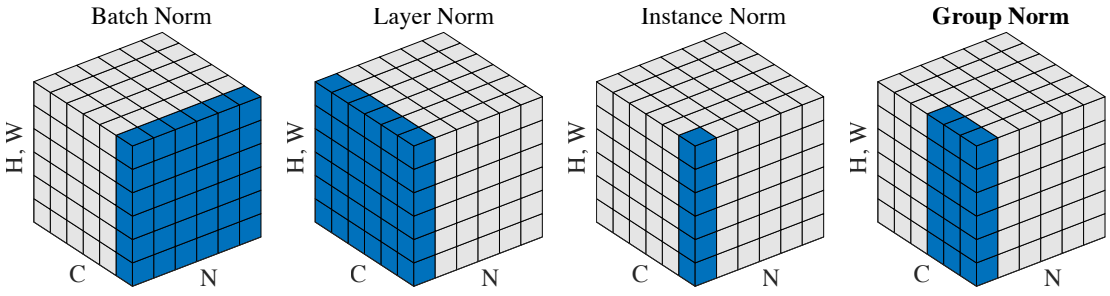


Figure 1.2 – The dimensions along which various normalization layers operate (Wu and He, 2018). N corresponds to the batch dimension, H,W,C are height, width, and channels (for image inputs).

Residual Connections While not technically a layer, we highlight the importance of residual connections to modern architectures, as popularized by He et al. (2015a).

These do not directly change the expressive power of a model, but similarly to normalization layers make optimization of deeper models significantly easier.

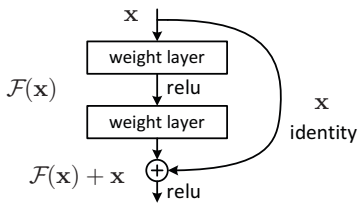


Figure 1.3 – An example of a residual block (He et al., 2015a)

Initializations An often overlooked but critical aspect of layers is the values used to initialize them when training with SGD. Naively setting each layer's weights from independent random distributions can lead to difficulty in training convergence and gradient signal propagation depending on the nonlinearities used between layers; these issues are generally resolved through careful design of initializations to ensure stability of signal propagation through layers (Glorot and Bengio, 2010; He et al., 2015b). Remarkably, appropriate design of initialization schemes can enable training of models with 10,000 layers, using neither residual connections nor normalization (Xiao et al., 2018).

1.2.2 Generalization

A critical property of a trained model is its ability to *generalize* to samples beyond the training set (though from the same distribution). Remarkably, deep learning models, in spite of being heavily overparameterized and capable of memorizing completely noisy data (Zhang et al., 2016), in practice have low **generalization gap** (the difference between error on training data and held-out testing data), and do not learn via memorization on structured data (Arpit et al., 2017). Unfortunately, we currently lack theory to convincingly explain these observations; in practical settings most generalization bounds are vacuous (greater than 1), and in many cases generalization measures derived from them do not even correlate with the generalization gap across architectures and training settings (Jiang et al., 2019; Dziugaite et al., 2021).

1.3 Self-Supervised Learning

A significant limitation of standard ML methods has been their reliance on explicit supervisory signal in the form of labels. This is a bottleneck, as modern deep learning often requries vast amounts of labelled data for competitive performance, and yet labelling introduces a dependency on costly and error-prone human annotation. The subfield of **self-supervised learning** (SSL) is focused on constructing novel objectives that exploit the rich inherent structure present in unlabelled data by itself. For example, simple tasks such as next-token prediction have driven massive advances in NLP, and are central to current state of the art models (Brown et al., 2020a). Pretraining with SSL objectives produces models which can transfer zero-shot to new tasks (Brown et al., 2020a; Radford et al., 2021), or can efficiently be finetuned with very few samples from the new task (Chen et al., 2020b; Grill et al., 2020a; Hénaff et al., 2019). In the latter setting, SSL pretraining is a form of learned initialization, in contrast to random initializations discussed earlier, and is reminiscent of unsupervised layer-wise pretraining that was historically critical to the training of deep neural networks (Erhan et al., 2010). For a more in-depth overview of recent advances in self-supervision, one can read Weng (2019) and Weng (2021).

1.3.1 Contrastive Learning

A recently successful approach to unsupervised representation learning has been **Contrastive Learning**, which seeks to maximize the latent-space similarity of positive pairs of examples and the latent-space *dis*similarity of negative pairs of examples. Positive pairs are generally different views of the same data sample, often using augmentation to obtain these separate views as in SwAV (Caron et al., 2020) or through image-text pairs scraped from the internet as in CLIP (Radford et al., 2021). Contrastive learning is effectively a classification problem, of identifying a positive sample in comparison to all other samples in a batch (which are negative samples), and can be referred to as contrastive instance discrimination. A specific instantiation of a contrastive loss looks like the following instance discrimination

objective from SimCLR (Chen et al., 2020b).

$$\text{NTXent}_{i,j} = -\log \frac{\exp\left(\frac{\mathbf{z}_i^\top \mathbf{z}_j}{\|\mathbf{z}_i\|\|\mathbf{z}_j\|\tau}\right)}{\sum_{k=1}^{2N} \mathbb{1}_{[k\neq i]} \exp\left(\frac{\mathbf{z}_i^\top \mathbf{z}_k}{\|\mathbf{z}_i\|\|\mathbf{z}_k\|\tau}\right)}$$

This represents the loss for the positive pair i,j within a batch of original size N (and $2N$ after producing a positive pair for each sample in it), with temperature scaling τ; the loss for a batch is the sum over the losses for each positive pair. Note that the terms inside exponentiations are merely cosine similarities between low-dimensional representations $\mathbf{z}$ for each input.

Contrastive objectives have been analyzed in great detail due to their success. Wang and Isola (2020) found that it is implicitly composed of a uniformity objective which (asymptotically) uniformly covers the latent space, and an alignment objective which ensures closeness of features from positive pairs. Zimmermann et al. (2021) showed that under certain assumptions the contrastive objective actually inverts the generative model of the data it is trained on. Cai et al. (2020) showed that a small minority of negatives (referred to as hard negatives) within a batch were both necessary and sufficient for optimal performance on downstream tasks, suggesting better strategies for negative sample selection could greatly reduce the computational requirements of contrastive learning methods.

Contrastive Learning suffers from certain drawbacks. The reliance on large numbers of negative samples requires huge batch sizes that introduce computational challenges (Chen et al., 2020b). Recently Grill et al. (2020b) demonstrated that negative samples could be entirely done away with, and that optimizing for alignment of positive pairs alone could be done in a stable manner with certain additional tricks such as an exponential moving average network providing regression targets for representations under different augmentations. Nevertheless, this approach and most other contrastive methods are reliant on heavy augmentation for positive samples, introducing representational invariance which can be harmful for downstream tasks (Xiao et al., 2021); such augmentations can also be hard to design in novel domains beyond images.

1.4 (Deep) Reinforcement Learning

The previous sections have focused on settings in which a fixed amount of data is provided to a learning algorithm, and for which the test data comes from the same distribution as this training data. These assumptions are poor when considering the manner in which natural intelligence learns – from an online and varied stream of data, and with the ability to interact with the environment and affect the distribution of new data being learned from and evaluated on. **Reinforcement Learning** (RL) is focused on this paradgim, of an agent learning online from feedback in an environment that they themselves affect. Specifically, agents are trained to maximize their expected *reward* from this environment. The formalisms that underpin this learning setting are briefly described as follows; more comprehensive treatments can be found in Sutton and Barto (2018) and Achiam (2018).[4]

1.4.1 Markov Decision Processes

At a timestep t, consider that we can fully describe the environment (including the agent in it) with a discrete or continuous **state** s_t, and an agent chooses to take a discrete or continuous action a_t (possibly including a choice to take no action). A scalar function $R(s_t, a_t, s_{t+1})$ determines the reward an agent receives from the environment when taking actions. In a **Markov Decision Process** (MDP, Bellman, 1957), we make the Markov assumption that the current state s_t contains all information necessary to model the dynamics of the environment, i.e. that **transitions** between states in the environment are completely modelled by a distribution $p(s_{t+1} \mid s_t, a_t)$. The important detail in this assumption is that prior states and actions are unnecessary for modelling transitions; the distribution of the next state depends only on the current state and action taken from it. An interaction sequence $s_t, a_t, r_t, s_{t+1}, a_{t+1}, \ldots$ is called a **trajectory**; when there are terminal states in the MDP (such as a "game over"), trajectories are finite and can be referred to as **episodes**. Finally, the metric of primary interest for an agent is the **return** $G = \sum_t r_t$ – the total reward obtained over an episode (such as the score over a game). A slightly modified objective is almost always used instead – the discounted sum of rewards using a geometric discounting factor $\gamma \in [0, 1]$: $G_\gamma = \sum_t \gamma^t r_t$; this

4. The background sections of Schwarzer (2020) are also relevant, as the RL article presented in this thesis builds on the method presented in that work.

discounting biases an agent towards prioritizing earlier rewards, and also ensures for unbounded trajectories that the return is a bounded quantity (for $\gamma < 1$).

1.4.2 Policies and Q-Learning

RL is concerned with finding optimal behaviours, that maximize expected return within MDPs. Agents do not necessarily know the structure of the MDP (the transition dynamics p, or the reward function R); and in real-world settings these are unknown to algorithm designers as well. Formally, agent behaviour can be defined by a **policy** $\pi(a \mid s)$, which is a distribution over actions given the current state. In settings with discrete actions, one popular approach to constructing a policy is indirectly via **Q-Learning**. The objective in this setting is to learn a (possibly non-unique) function $Q^*(s, a)$ which quantifies the expected return when taking action a in state s, and from there on acting optimally; this provides a measure of the quality of a state-action pair. Acting optimally is defined recursively: an optimal policy π^* acts greedily and takes the highest-valued action according to Q^* in every state: $\pi^*(s) = \arg\max_a Q^*(s, a)$. Under any policy π (optimal or not), the associated Q^π-function adheres to a very useful recursive decomposition known as the **Bellman expectation equation**:

$$Q^\pi(s_t, a_t) = \mathop{\mathbb{E}}_{s_{t+1} \sim p}[R(s_t, a_t, s_{t+1}) + \gamma \mathop{\mathbb{E}}_{a_{t+1} \sim \pi}[Q^\pi(s_{t+1}, a_{t+1})]]$$

Under the optimal policy which acts greedily, this can be further simplified to the **Bellman optimality equation**:

$$Q^*(s_t, a_t) = \mathop{\mathbb{E}}_{s_{t+1} \sim p}[R(s_t, a_t, s_{t+1}) + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1})]$$

One of the central advances in modern RL has been the use of neural networks as powerful function approximators; with **Deep Q-Networks** (DQN, Mnih et al., 2015), the Q_θ-function is parameterized by a neural network with weights θ. To learn an approximation to Q^*, we use standard gradient-based learning to minimize the **Temporal Difference** (TD) error: the discrepancy between the left and right sides of the Bellman optimality equation under the model Q_θ.

The use of neural networks for Q-learning suffers from inherent instabilities (van Hasselt et al., 2018), and a variety of additional methods are commonly used to

mitigate this problem – the Rainbow DQN implementation combines many of these improvements and provides a strong baseline to build off of (Hessel et al., 2018).

1.4.3 Data-Efficient Atari

One of the most widely used benchmarks in Deep RL has been the **Atari Learning Environment** (Bellemare et al., 2013), a suite of 57 video games from the Atari 2600 console. As Atari games were designed to be played by humans, they also provide a benchmark for comparison to human *data-efficiency*. In Mnih et al. (2015), human testers were given two hours (roughly 100K environment steps) to learn to play each game before evaluation on them. In comparison, DRL algorithms are often trained with months or years of human-equivalent interaction (Badia et al., 2020; Campos et al., 2021), a clearly unsustainable trend if DRL is to see usage on problems of greater complexity in the physical world we inhabit. The Atari-100K benchmark limits DRL interaction to 100K interactions on each of a subset of 26 Atari games, as a way to measure progress on the efficiency of RL algorithms. This benchmark has driven considerable progress on data-efficiency, with varied approaches such as the use of reconstruction-based models for asynchronous training (Kaiser et al., 2019), better tuning of hyperparameters for standard algorithms to trade-off interaction for computation (van Hasselt et al., 2019), the use of data augmentation (Kostrikov et al., 2021), and the use of self-supervised auxiliary objectives (Schwarzer et al., 2021a). Nevertheless, no method has achieved human-level sample efficiency on all the games of this benchmark as yet.

2 Does Your Model Memorize Where You Think It Does?

Authors Nitarshan Rajkumar[1,2], David Krueger[1,2], Laurent Charlin[1,3]

Affiliation [1]Mila, [2]Université de Montréal, [3]HEC Montréal

Abstract Memorization in neural networks has generally been investigated by training models on artificially "noisy" or mislabelled data, a setting known to be harmful for generalization. By contrast, memorization as encountered in practice, on mostly noiseless ("clean") long-tailed data, has been shown to be *beneficial* for generalization. In this paper, we conduct an empirical study into these forms of memorization and find significant differences in where they occur *within* models. In particular, we find that memorization of clean data in convolutional models generally occurs in proportion to *convolutional parameter count*, and does not necessarily occur in deeper layers. By contrast, memorization of noisy data is largely a factor of depth, predominantly occurring in the final few layers of a network.

Contributions The idea for this work, all experimentation, and all writing was done by myself. David provided feedback and helped edit the paper. Laurent similarly helped with editing, and supervised this work.

Submission This is ongoing work, and as presented has been submitted to the ICML 2021 Workshop on Overparameterized Models where it is under review.

2.1 Introduction

Overparameterized neural networks trained with stochastic gradient descent (SGD) are capable of fitting, or memorizing, completely noisy data (Zhang et al., 2016). This observation laid bare the failure of classical theories of generalization

to explain the learning abilities of neural networks, and Arpit et al. (2017) further advocated that such explanations must accordingly be data-dependent.

These works generally focused on **noisy memorization** – i.e. on (artificially) corrupted or mislabelled samples. Noisy memorization can induce worse generalization and robustness (Liu et al., 2019; Maennel et al., 2020; Sanyal et al., 2020), but experimentally this setting is appealing as it is easy to artificially intervene on noise and control the amount of memorization needed to fit a dataset.

The memorization we encounter in practice however, and which contributes to a generalization gap, is largely in the form of **clean memorization** – i.e. on samples which are not mislabelled or corrupted, but are relatively atypical with respect to their classes. This form of memorization has recently been shown to be theoretically and empirically *beneficial* for generalization on minority subgroups when data is drawn from long-tailed distributions, as is common for natural data (Feldman, 2019; Feldman and Zhang, 2020).

Analysis of these two forms of memorization has until now occurred separately, with sometimes conflicting results. For example, prior work on noisy memorization has claimed that memorization emerges as a property of model depth (Cohen et al., 2019; Stephenson et al., 2021), while a limited experiment in Feldman and Zhang (2020) suggested that clean memorization barely occurs in the last layer of a model. Accordingly, the structural similarities, or dissimilarities, of noisy and clean memorization are poorly understood.

We take a simple, empirical approach to understanding memorization of clean and noisy data, using standard architectures, training procedures, and natural data. In particular, we aim to attribute such memorization to specific layers *within* a model. By explicitly drawing a distinction in analysis between clean and noisy memorization, we demonstrate the heterogeneous, architecture-and-data-dependent nature of where memorization occurs. We show that clean memorization does not *necessarily* occur in deep layers, and negligibly occurs in final linear layers. This is in stark contrast to noisy memorization, which predominantly occurs in the deepest layers, including the final linear layers of convolutional models. Surprisingly, for these convolutional models, we find that clean memorization generally occurs in linear proportion to *convolutional* parameter count. This finding suggests differing roles for capacity and depth in clean and noisy memorization.

2.2 Methodology

Defining Memorization We focus our attention on **memorization-required samples** – i.e. samples which are correctly classified when included in the training dataset, and incorrectly classified if held out of the training process, following the definition of memorization in Feldman and Zhang (2020). Unlike previous notions of memorization, this definition doesn't require artificially injecting noise in the data, and can identify clean but atypical data points as memorized.

To attribute memorization to individual layers, we use a *set* of memorization-required samples. At the extremes of inclusion or exclusion from the training process, a model will have 100% or close to 0% accuracy on this set, respectively. In between these extremes, we consider what would happen if only the first l layers were trained with this set held-in. For how many of the samples would the l'th layer learn a sufficient representation, such that the remaining layers could train without these samples and still classify them correctly? We draw inspiration from the memorization estimator of Feldman and Zhang (2020) to answer this question, and describe our method in detail below.

Partial Retraining with Memorization-Required Samples Held-out

1. **Pretrain a Model.** Train an L-layer model h on the full training set $\mathcal{D}_T$, using algorithm $\mathcal{A}$: $h \leftarrow \mathcal{A}(\mathcal{D}_T)$.
2. **Hold Out Memorization-Required Samples.** Separate the training set $\mathcal{D}_T$ into two disjoint sets:
   (a) a set $\mathcal{D}_M$ containing only memorization-required samples
   (b) a set $\mathcal{D}'_T$ containing all other samples
3. **Partially Retrain On Non-Memorized Samples.** For $l \in [0, L)$:
   (a) Initialize the first l layers of an L-layer model h'_l with the first l layers of h, and **freeze** these layers. Initialize the remaining layers of h'_l randomly.
   (b) Using the first l layers of h'_l as a fixed feature extractor, train the remaining $L-l$ layers of h'_l on $\mathcal{D}'_T$, using the same algorithm used to train h: $h'_l \leftarrow \mathcal{A}^h_l(\mathcal{D}'_T)$ (h'_0 corresponds to retraining the entire model on $\mathcal{D}'_T$, and h'_L is equivalent to h)
   (c) Repeat the above step over multiple random seeds.

4. **Calculate Local Memorization.** For each layer l, using the partially retrained models from the previous step, calculate **local memorization** as the expected difference in classification accuracy on $\mathcal{D}_M$ between when layer l is frozen (i.e. was pretrained on both $\mathcal{D}'_T$ and $\mathcal{D}'_M$) and not (i.e. was retrained only on $\mathcal{D}'_T$):

$$\mathop{\mathbb{E}}_{h'_l \leftarrow \mathcal{A}^h_l(\mathcal{D}'_T)} \left[\mathtt{acc}\left(h'_l, D_M\right)\right] - \mathop{\mathbb{E}}_{h'_{l-1} \leftarrow \mathcal{A}^h_{l-1}(\mathcal{D}'_T)} \left[\mathtt{acc}\left(h'_{l-1}, D_M\right)\right]$$

Intuitively, local memorization is the marginal contribution of a layer to the (100%) accuracy of the full model on the set of memorization-required samples. We emphasize that this process does *not* identify where specific samples are memorized, and instead measures memorization in aggregate over a set of samples.

Datasets and Architectures All experiments are done using CIFAR-100 (Krizhevsky, 2009a). We consider variants of three architectures for image classification: VGGs (Simonyan and Zisserman, 2015), ResNets (He et al., 2015a), and Vision Transformers (ViTs) (Dosovitskiy et al., 2020). We slightly adapt these models for the setting of CIFAR-100: for VGGs we remove the average pool, and change the first linear layer to be of size 512x4096; for ResNets we replace the first convolutional layer (7x7 kernel, 2x2 stride) with a smaller one (3x3 kernel, 1x1 stride), and remove the subsequent max pool; for ViTs we use the recommended modifications of Hassani et al. (2021). For VGGs, we define a layer to be a convolutional or linear layer combined with the batch normalization that follows; for ResNets, we treat residual blocks as single layers; for ViTs, we define layers to be either residual self-attention or linear layers combined with their preceding normalization layer.

Constructing Memorization-Required Subsets We make use of the memorization scores provided by Feldman and Zhang (2020) as an oracle for identifying 5756 memorization-required samples (roughly 11.5% of the training set) in CIFAR-100, which we include in our holdout set used for measuring clean memorization. Some of these samples are in fact mislabelled (Song et al., 2020; Pleiss et al., 2020) – technically noisy according to our terminology – but we assume they are minimal in quantity and include them in this set of points for ease of analysis. For measuring noisy memorization, we simply shuffle the labels in this same set of samples (even when they are held in during pretraining).

Table 2.1 – Summary of models used.

Model	Depth	Params	Test Accuracy
VGG11	11	28.5M	66.6 / 63.5
VGG13	13	28.7M	70.2 / 67.3
VGG16	16	34.0M	69.8 / 66.2
VGG19	19	39.3M	69.8 / 65.2
RN18	10	11.2M	72.0 / 67.9
RN34	18	21.3M	73.7 / 68.6
RN50	18	23.7M	74.1 / 68.7
WRN50	18	67.0M	74.6 / 69.4
RN101	35	42.7M	73.8 / 68.6
ViT7/4	16	3.74M	60.6 / 56.7

Training We follow the training procedure of Feldman and Zhang (2020) across all experiments: training for 160 epochs, SGD with a learning rate of 0.4 and momentum of 0.9, linear learning rate warmup from 0 over the first 15% of epochs and linear decay back to 0 for the remaining epochs, a batch size of 512, and random crop and horizontal flip data augmentations. All models (and their partially retrained descendants) are trained to convergence and reach over 99.9% training accuracy; test accuracies (when trained including either clean or noisy subsets) are presented in Table 2.1. All models except the ResNet-101 are trained with mixed-precision. For partial retraining we average results over 5 random seeds for clean memorization experiments; we found little variance between measurements across these seeds, and used 1 random seed for noisy memorization experiments due to compute constraints.

2.3 Results

We present our main results in Figure 2.1, which shows the distribution of local memorization on clean and noisy data across the 10 models considered. We imme-

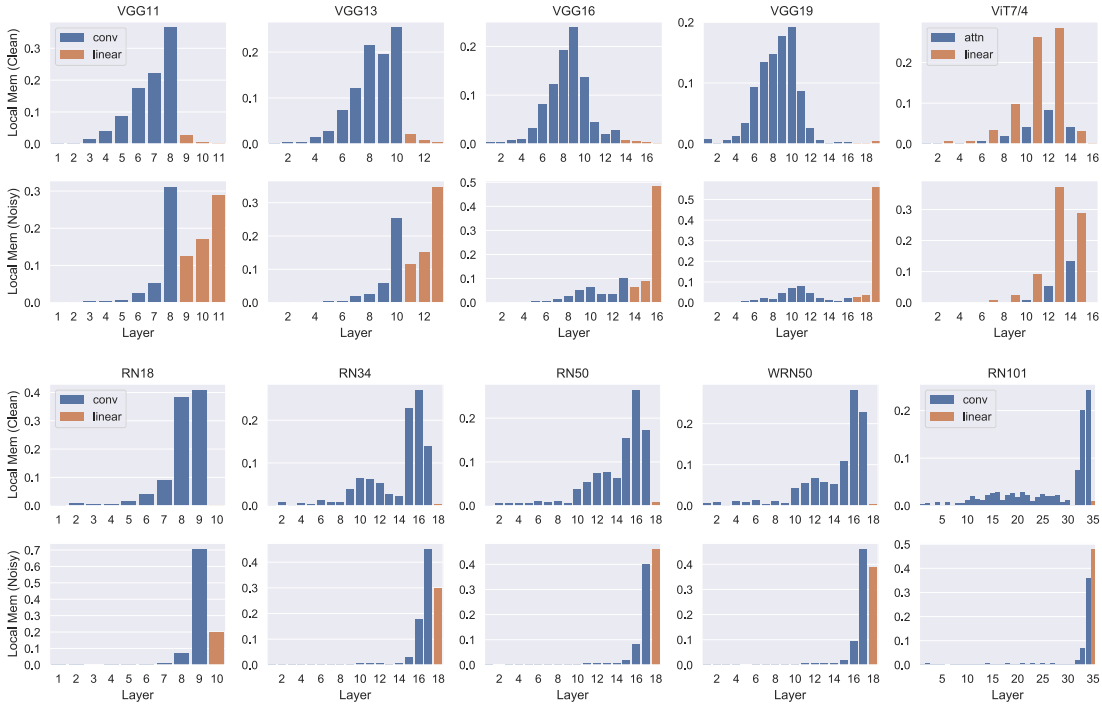
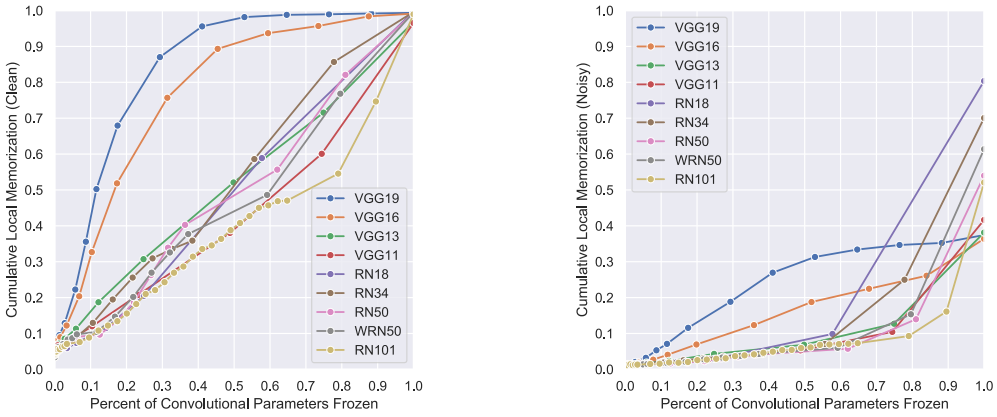


Figure 2.1 – Clean and noisy memorization occur in different layers. Each plot presents local memorization on clean (first row) and noisy (second row) data, across all models evaluated (column titles, see Table 2.1). Additional visualizations of parameter count and layer input size are presented in Figure 2.3.

diately observe that for both clean and noisy memorization, there is no consistent behaviour across all the models evaluated.

We also visually verify from Figure 2.2a that the clean memorization subset was being almost entirely memorized – when these samples are included in the training set, accuracy on this subset is nearly 100%, but when they are withheld completely from training the subset accuracy drops to ∼5% (as CIFAR-100 has 100 classes, random selection is 1%).

Clean memorization does not necessarily occur in deeper layers. We can see from Figure 2.1 that for convolutional models, the final linear layers contribute negligibly to clean memorization on both VGG and ResNet architectures, extending the findings of Feldman and Zhang (2020). This is not a property of the linear layers themselves, but their position at the end of the network; intermediate linear layers are responsible for the vast majority of clean memorization in the Vision



(a) Clean Memorization Subset

(b) Noisy Memorization Subset

Figure 2.2 – Clean memorization occurs generally linearly with convolutional parameter count. Plots present accuracy on heldout memorization-required subsets, with respect to percent of convolutional parameters frozen under the retraining procedure outlined in Section 2.2. Further visualizations are presented in Figure 2.4 and Figure 2.5.

Transformer (despite being interleaved with self-attention layers of similar parameter count). We also make note of the manner in which clean memorization occurs in VGGs – the layers in which this occurs are roughly the same (6-12) even as depth is increased from VGG13 to VGG19, with negligible memorization occurring in the additional convolutional layers of the VGG19.

In contrast to clean memorization, noisy memorization occurs largely in later layers, including linear layers. Figures 2.1 and 2.2b show that noisy memorization for convolutional models predominantly occurs in the final convolutional layer and the subsequent linear layers (for VGGs this is where the majority of parameters are). These findings align with prior work investigating noisy memorization and finding that it abruptly occurs at the latest layers which specialize to fit this data (Cohen et al., 2019; Maennel et al., 2020; Stephenson et al., 2021). Interestingly, with increasing depth noisy memorization moves largely to the final linear layer alone across VGGs and ResNets, in contrast to clean memorization which either stays similarly distributed (for ResNets) or moves relatively earlier in the network (for VGGs). This insight could be used for targeting the final layers responsible for noisy memorization in a network (with retraining or perhaps for identifying mislabelled examples), with minimal impact on earlier layers responsible for more

helpful clean memorization.

Clean memorization generally occurs in proportion to convolutional parameter count. In Figure 2.2a, we observe that *all* ResNets exhibit a remarkably linear relationship between the percent of convolutional parameters frozen and the accuracy on the clean memorization heldout subset. The smaller VGG models also exhibit this relationship, though increased depth in the VGG16 and VGG19 leads to this breaking down and memorization occurring relatively earlier in the convolutional layers of the network. This difference between the ResNets and VGGs seems to indicate the importance of residual connections for evenly allocating network capacity to clean memorization. Furthermore, the contrast between clean memorization being proportional to capacity and noisy memorization being a factor of depth is one we believe to be unexplored in literature as yet.

2.4 Conclusion

In this work, we introduced a new approach to quantifying local memorization within neural networks, which we then applied across a variety of models to show the architecture-dependence of local memorization. Furthermore, we have used this approach to demonstrate the heterogeneous, data-dependent nature of local memorization, by contrasting between clean and noisy memorization, and their respective relations with capacity and depth. Our findings contribute to developing the understanding of memorization in neural networks; they also suggest that analysis of noisy memorization may lead to conclusions which are not applicable to the clean memorization setting closer to what is encountered in practice.

2.4.1 Future Work

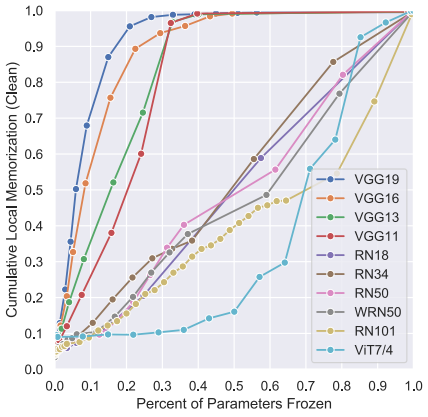
The primary limitation of our work is that it views memorization at the macro level over a subset of data, and does not provide insight into the stochasticity of memorization at a sample-level as Feldman and Zhang (2020) did. Furthermore, by treating entire residual blocks as layers we sidestepped the branching nature of residual connections as well as the effect of downsampling in ResNets. We intend

to closely investigate sample-level localized memorization as well as architectural factors in greater detail.

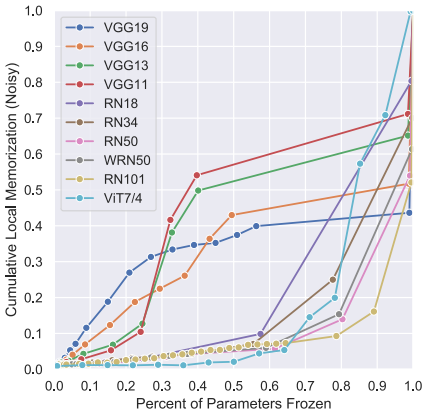
2.5 Additional Plots

In this section we provide additional plots that visualize properties such as parameter count and input size for all layers of all models considered, as well as exhaustive plots that show the relationship between cumulative memorization and parameter count or layer depth.

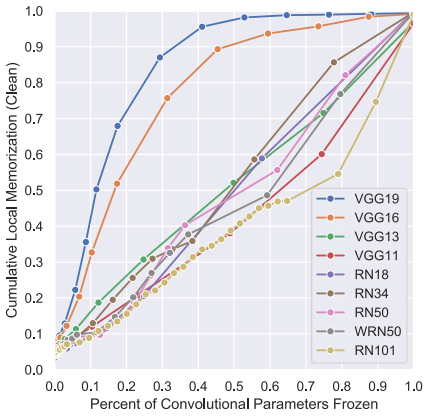
Figure 2.3 – Summary metrics for all models examined (column titles, see Table 2.1). *First Row:* Local memorization on clean data. *Second Row:* Local memorization on noisy data. *Third Row:* Per-layer contribution to total parameter count. *Fourth Row:* Per-layer input size.



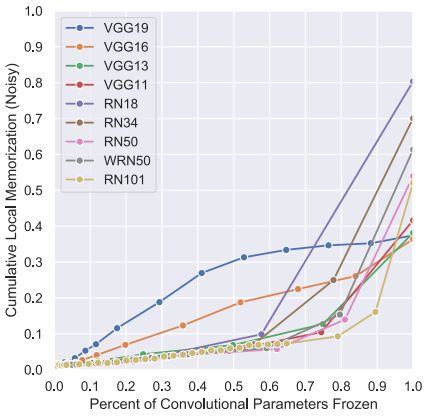
(a) Clean memorization w.r.t. % of parameters frozen.



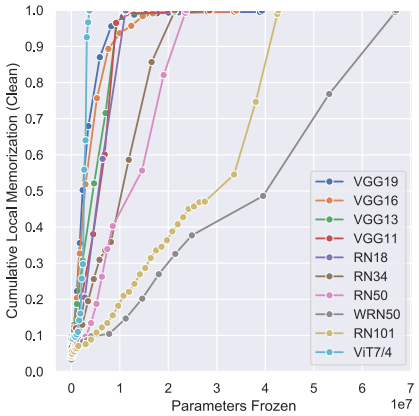
(b) Noisy memorization w.r.t. % of parameters frozen.



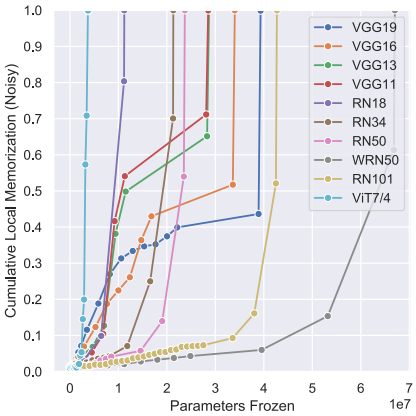
(c) Clean memorization w.r.t. % of convolutional parameters frozen.



(d) Noisy memorization w.r.t. % of convolutional parameters frozen.

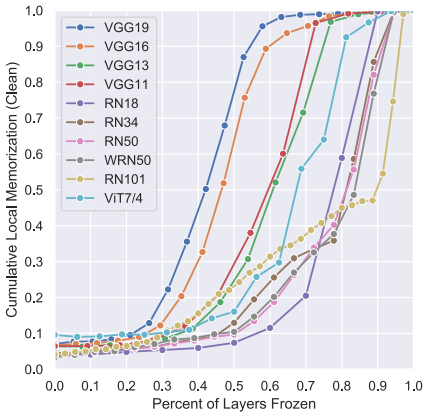


(e) Clean memorization w.r.t. # of parameters frozen.

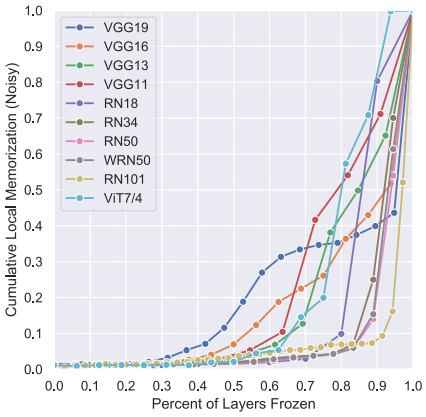


(f) Noisy memorization w.r.t. # of parameters frozen.

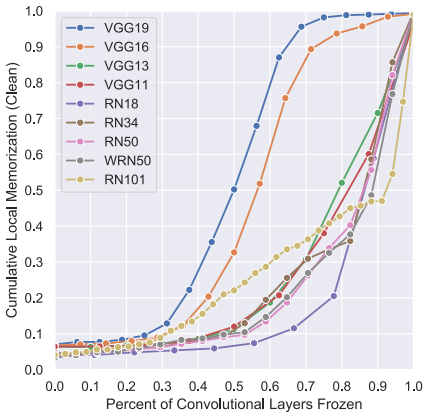
Figure 2.4 – **Parameter-wise** analysis of cumulative clean and noisy local memorization, under the retraining procedure of Section 2.2.



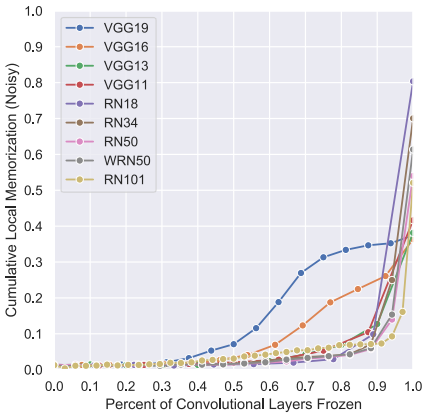
(a) Clean memorization w.r.t. % of layers frozen.



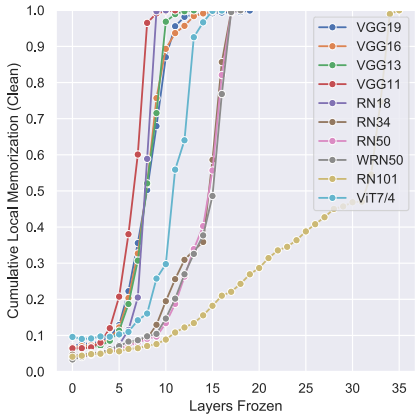
(b) Noisy memorization w.r.t. % of layers frozen.



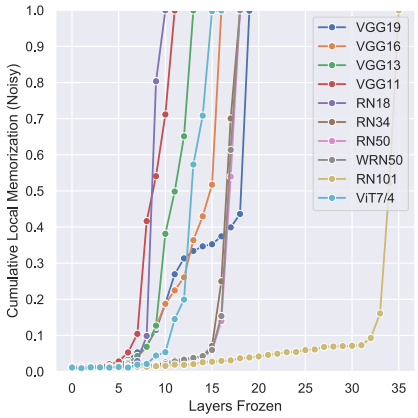
(c) Clean memorization w.r.t. % of convolutional layers frozen.



(d) Noisy memorization w.r.t. % of convolutional layers frozen.



(e) Clean memorization w.r.t. # of layers frozen.



(f) Noisy memorization w.r.t. # of layers frozen.

Figure 2.5 – **Depth-wise** analysis of cumulative clean and noisy local memorization, under the retraining procedure of Section 2.2.

3 Instance-Attribution through the Lens of Self-Supervision

Authors Nitarshan Rajkumar[1,2], Eeshan Gunesh Dhekane[1,2], David Krueger[1,2], Laurent Charlin[1,3]

Affiliation [1]Mila, [2]Université de Montréal, [3]HEC Montréal

Abstract Several methods have been proposed towards understanding the structural regularity of datasets used to train deep learning models, and on attributing model decisions to these training samples. Unfortunately these methods tend to have significant computational requirements that hinder their applicability to the massive models and datasets used in practice. Instead of introducing any new methods, we explore how existing methods behave under explicit separation of task-specific learning and representation learning – specifically by using linear classifiers and pretrained self-supervised encoders. We demonstrate how this decoupled approach produces measurements of sample *consistency*, *memorization*, and *influence* that are highly correlated with the corresponding values produced by fully-supervised equivalents, while requiring four orders of magnitude less compute. We also analyze where this approach behaves differently – in particular on the long-tail of data, where we revisit the claims of the Long-Tail Theory (Feldman, 2019).

Contributions The idea for this work, all experimentation, and almost all writing was done by myself. Eeshan Dhekane provided assistance with writing and editing. David provided feedback on the project direction and experiments. Laurent provided feedback and supervision throughout this project.

Submission This is ongoing work and has not been submitted anywhere.

3.1 Introduction

Interpretability via sample importance Deep learning is being increasingly employed on highly sensitive and safety-critical tasks such as face recognition (Buolamwini and Gebru, 2018), medical imaging (Selvikvåg Lundervold and Lundervold, 2018), and autonomous driving (Huang and Chen, 2020). Underlying objectives in such settings can be hard-to-formulate, involving factors such as unbiasedness, ethicality, and legality (Lipton, 2016). Thus, it is important to be able to interpret the decisions made by these models to allow for human evaluation with respect to these factors. Measuring the importance of individual training samples to the training procedure and test decisions is one promising approach to interpretability. Measures of **sample importance** include consistency (Jiang et al., 2020) and memorization (Feldman and Zhang, 2020), which measures the regularity of a sample with respect to the rest of the dataset, and influence (Koh and Liang, 2017), which measures the contribution of a training sample's presence to the correct classification of a test sample.

Challenges when scaling Unfortunately, dataset-focused approaches to interpretability tend to have extreme computational requirements when applied to the large datasets and models used in practice. For instance, estimating influence and memorization in Feldman and Zhang (2020) and consistency in Jiang et al. (2020) required nearly 1 *million* GPU-hours when applied to ResNet50's on ImageNet. As another example, the influence estimators proposed by Pruthi et al. (2020) and Koh and Liang (2017) require Jacobian and Hessian computations respectively, which are computationally demanding or entirely infeasible for standard model architectures with millions or billions of parameters. Consequently, some form of approximation is generally necessary for practical use, though potentially at significant cost to accuracy. Koh and Liang (2017) made use of approximate inverse-Hessian Vector product methods (Agarwal et al., 2017), though recent work has shown that for deep models this contributes to erroneous and low quality influence estimates (Basu et al., 2020). Pruthi et al. (2020) restricted calculations of the Jacobian and Hessian only to the final layer of the model, though Feldman and Zhang (2020) and Rajkumar et al. (2021) demonstrated that a form of memorization useful for classifying rare test samples largely occurs in the non-final layers.

Decoupling representation and task The importance of a training sample has two components: **i.** an aspect dependent solely on the regularity of the unlabelled part with respect to the rest of the data, and **ii.** the label-dependent aspect, with respect to a particular task. While fully-supervised models trained end-to-end (E2E) implicitly capture both of these components, the former data-dependent component could be entirely captured through a strong unsupervised representation. Simultaneously, the label-dependent component could be fully isolated to a significantly smaller model operating on top of such representations. Self-supervised learning (SSL) provides a paradigm under which such a decoupling is possible; recent methods learn representations which can be fit with linear classifiers to match the performance of fully-supervised models on challenging computer vision benchmarks (Chen et al., 2020a; Grill et al., 2020b; Caron et al., 2020). Furthermore, recent analysis has demonstrated that models trained in this manner behave similarly to E2E models (Geirhos et al., 2020). In this paper we draw inspiration from these advances and evaluate if and how self-supervised encoders combined with linear classifiers can be used to efficiently estimate measures of sample importance; we refer to this decoupled setting as SSS for Self-Supervised + Simple classifier [1])

The rest of this paper is structured as follows. In Section 3.2 we introduce notation and background material on sample importance. In Section 3.3 we present results comparing measures calculated via SSS and E2E approaches; we find a high level of correlation in produced scores, while reducing computational requirements by four orders of magnitude when using SSS. In Section 3.4 we conclude with a brief investigation into the importance of memorization for test performance in the SSS setting, finding further evidence in support of the Long-Tail Theory of Feldman (2019).

3.2 Background

How is training and inference of a model affected by any single training sample? Different metrics for sample importance estimation attempt to quantify this

1. Terminology borrowed from Bansal et al. (2020)

notion; we introduce the three metrics studied in this paper, broadly inspired by the treatments of Feldman and Zhang (2020) and Jiang et al. (2020).

Notation Formally, we are interested in the predictions of learned predictors h obtained by training a stochastic learning algorithm $\mathcal{A}$ on a dataset $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$ with N samples. We further define $D^{\setminus k}$ to be the dataset obtained by removing the k-th sample from D.

The **influence** score of training sample i drawn from dataset D on an arbitrary test sample j (not necessarily in D) is the expected change in the classification accuracy of sample j when sample i is removed from the training set:

$$\texttt{infl}\left(D, \mathcal{A}, i, j\right) = \underset{h \leftarrow \mathcal{A}(D)}{Pr}\left(h(\mathbf{x}_j) = y_j\right) - \underset{h \leftarrow \mathcal{A}(D^{\setminus i})}{Pr}\left(h(\mathbf{x}_j) = y_j\right) \tag{3.1}$$

Of the three measures we use, influence is the only one which goes beyond the training set and can make use of test data.

The **memorization** score is straightforwardly defined as self-influence; i.e. the expected change in classification accuracy on a sample i itself when it is removed from the training set:

$$\texttt{mem}\left(D, \mathcal{A}, i\right) = \texttt{infl}\left(D, \mathcal{A}, i, i\right) \tag{3.2}$$

Finally, the **consistency** score of sample i is the expected classification accuracy of a learned model h on sample i, when sample i itself is held out of the training dataset:

$$\text{cons}\left(D, \mathcal{A}, i\right) = \underset{h \leftarrow \mathcal{A}(D^{\setminus i})}{Pr}\left(h(\mathbf{x}_i) = y_i\right) \tag{3.3}$$

Intuitively, consistency measures how well supported a sample is by the rest of the data distribution. Note that this is merely the second term in the formula for memorization; consistency is directly proportional to memorization when classification accuracy on a training set is (nearly) 100%, as is common in the E2E setting **but not** in SSS.

Leave-M-In (LMI) Estimation A naïve approach to calculating these metrics is a leave-one-out strategy, which requires retraining models on each of the N pruned datasets $D^{\setminus k}$. As Feldman and Zhang (2020) pointed out, estimating these

metrics with standard deviation σ would require $\Omega(1/\sigma^2)$ training runs per training sample, for a total computational complexity of $\Omega(N/\sigma^2)$ to calculate measures for all samples in D. For the large datasets and models used in practice, this approach is entirely intractable.

Feldman and Zhang (2020) introduced an estimator, which we refer to as the Leave-M-In (LMi) estimator, that provides a tractable method to simultaneously compute consistency and influence estimates for *all* samples within a fixed training dataset and test dataset. It operates by measuring importance with respect to T subsets S_t of D, each of size M, that each have an associated model h_t trained on them; furthermore, it only requires $O(1/\sigma^2)$ training runs to estimate importance metrics within standard deviation σ.

$$\widetilde{\texttt{infl}}_M(D, \mathcal{A}, i, j) = \underset{t\sim\{1\ldots T\}}{P}(h_t(\mathbf{x}_j) = y_j \mid i \in I_t) - \underset{t\sim\{1\ldots T\}}{P}(h_t(\mathbf{x}_j) = y_j \mid i \notin I_t) \tag{3.4}$$

$$\widetilde{\texttt{cons}}_M(D, \mathcal{A}, i) = \underset{t\sim\{1\ldots T\}}{P}(h_t(\mathbf{x}_i) = y_i \mid i \notin I_t) \tag{3.5}$$

I_t represents the indices of training samples that were included in S_t. Feldman and Zhang (2020) balanced between estimator accuracy and compute requirements (for memorization and influence) by setting subset size $M = 0.7N$ and the number of trials $T = 2000$ for ImageNet-1K and $T = 4000$ for CIFAR-100; Jiang et al. (2020) used these same settings to calculate consistency.

Challenges with Further Approximation The LMI-estimator is tractable, yet nevertheless incurs significant computational cost; requiring 800,000 GPU-hours to calculate scores on ImageNet for Resnet-50's. Feldman and Zhang (2020) conducted a limited experiment to see if only retraining the last layer would be sufficient for calculating memorization and influence scores, finding that this was not the case. Furthermore the experiments and results in Chapter 2 demonstrate that memorization as exhibited on natural data can occur within networks in proportion with capacity for convolutional models; not including all these layers in an analysis of sample importance will therefore miss out on the contribution of these underrepresented long-tail samples.

3.3 Consistency and Influence via SSS

We start with the observation that current SSS methods approach or surpass the performance of equivalent E2E models on large-scale image classification tasks (Grill et al., 2020b; Chen et al., 2020a; Caron et al., 2020, 2021). Furthermore, in the SSS setting the linear classifier is the **only** component of the network in which a dependence on labels can occur. Thus, we calculate memorization, consistency, and influence scores using the LMI-estimator isolated to linear classifiers trained on top of large pretrained encoders. In this section, we first provide details on this methodology, and then compare the similarities and differences between scores computed by SSS and E2E approaches.

3.3.1 Methodology

Datasets We restrict our experiments to the image classification datasets used in Feldman and Zhang (2020) and Jiang et al. (2020); in particular, our primary experiments are on ImageNet-1K (Russakovsky et al., 2015). In Section 3.4 we also evalute on CIFAR-100 (Krizhevsky, 2009b), as well as ImageNetV2 (Recht et al., 2019), which is a new test set for ImageNet with subtle distribution shift that causes significant performance drops, and ciFAIR-100 (Barz and Denzler, 2020), a modification of the CIFAR-100 test set, with duplicates replaced with new images.

Self-Supervised Encoders We experiment with two state-of-the-art self-supervised methods: SwAV (Caron et al., 2020) and CLIP (Radford et al., 2021). SwAV is an online clustering algorithm which enforces cluster consistency between different views of images via a contrastive objective. CLIP is composed of an image and text encoder which are jointly trained in a contrastive manner on image-text pairs (we only make use of the pretrained image encoder from CLIP). We use pretrained encoders made publicly available by authors of these papers: SwAV provides pretrained encoders for a ResNet-50 pretrained on (unlabelled) ImageNet itself, and CLIP provides a Vision Transformer (ViT) pretrained on a private dataset scraped from the internet. [2]

2. While quite different from other self-supervised approaches due to its multimodal nature, we believe that the abundance of the paired data used to train CLIP justifies describing it as a self-supervised method.

Table 3.1 – Parameter count and ImageNet classification performance.

		Parameters		Accuracy		
Model	Dim.	Encoder	Classifier	Train	Test	V2
E2E-RN50	2048	—	27.1M	—	73%	—
SwAV-RN50	2048	25M	2.1M	81.6	73.2	—
SwAV-RN50-R2	4096	25M	4.1M	87.0	73.0	—
SwAV-RN50-R4	8192	25M	8.2M	91.7	71.3	—
CLIP-ViT	512	87M	0.5M	81.4	74.6	62.9

Linear Classifiers In training the linear classifiers, we use SGD with a learning rate of 0.3, with Nesterov momentum of 0.9, a batch size of 1024, and training is done over 40 epochs. No data-augmentation is used (though this was used when pretraining the encoders); this allows us to calculate representations for all images once, and then train the linear classifiers on just these stored representations, greatly speeding up training. For further speed we also employ mixed-precision training, at no effect to classification performance. We use the exact setup of Feldman and Zhang (2020) – we train 2000 classifiers, each on different 70% random subsets of the full training set. As seen in Table 3.1, each SSS model reaches test accuracies (when trained on a 70% subset) comparable to the E2E model accuracy reported in Feldman and Zhang (2020).

Widening Representations for Memorization As seen in Table 3.1, linear classifiers on the representations we obtain from SwAV (dimension 2048) and CLIP (dimension 512) do not reach 100% accuracy on the training set, due to limited capacity. To improve the quality of memorization estimates (relative to supervised models which can reach 100% training accuracy), we also run additional experiments where we widen the representation of the SwAV ResNet50 by a factor of 2 (-R2) or 4 (-R4). This is done by increasing the output size of that model's final average pooling layer; this does not change the number of parameters in the encoder at all.

Table 3.2 – Correlations between SSS and E2E Scores on ImageNet.

	Pearson r		Spearman ρ		Kendall τ	
Method	mem	cons	mem	cons	mem	cons
CLIP-ViT	0.35	0.64	0.50	0.67	0.38	0.53
SwAV-RN50	0.41	0.76	0.56	0.75	0.44	0.62
SwAV-RN50-R2	0.60	0.77	0.70	0.78	0.55	0.64
SwAV-RN50-R4	0.71	0.77	0.76	0.78	0.60	0.64

3.3.2 Consistency and Memorization

In Table 3.2 we present correlations between memorization and consistency scores calculated via SSS and those calculated using E2E models and provided by Feldman and Zhang (2020) and Jiang et al. (2020). We observe a strong correlation between consistency scores for both approaches, and across all models – the Pearson and Spearman correlations are consistently greater than 0.75 for the SwAV models, and greater than 0.64 for the CLIP model. Unsurprisingly, without representation widening there is a much weaker correlation across models with the memorization scores; wider representations improve this correlation. These results imply that we can recover a significant amount of information about the supervised estimates of consistency via our self-supervised approach.

To visuallly inspect these results, we first look at class-level regularity for dataset interpretability. As dataset designers we may wish to see how consistent samples within a class are, and whether a class requires further attention or even subdivision into more fine-grained classes. In Figure 3.1 we plot a class-level consistency density plot as was done in Jiang et al. (2020), with the x-axis being the mean consistency score for a class, and the y-axis being the standard deviation of scores within a class; points at the bottom right correspond to those classes which have high consistency scores for a large number of points without much variation. In addition, we plot 5 data points marked with $\star$ corresponding to the 5 classes identified in Jiang et al. (2020) as characteristic of the consistency structure of ImageNet-1K. The overall shapes of the consistency density plots match for the SSS and E2E approaches, and the relative orderings of the characteristic classes remain the same; implying that we can recover the class-level structure solely based on our approach in a

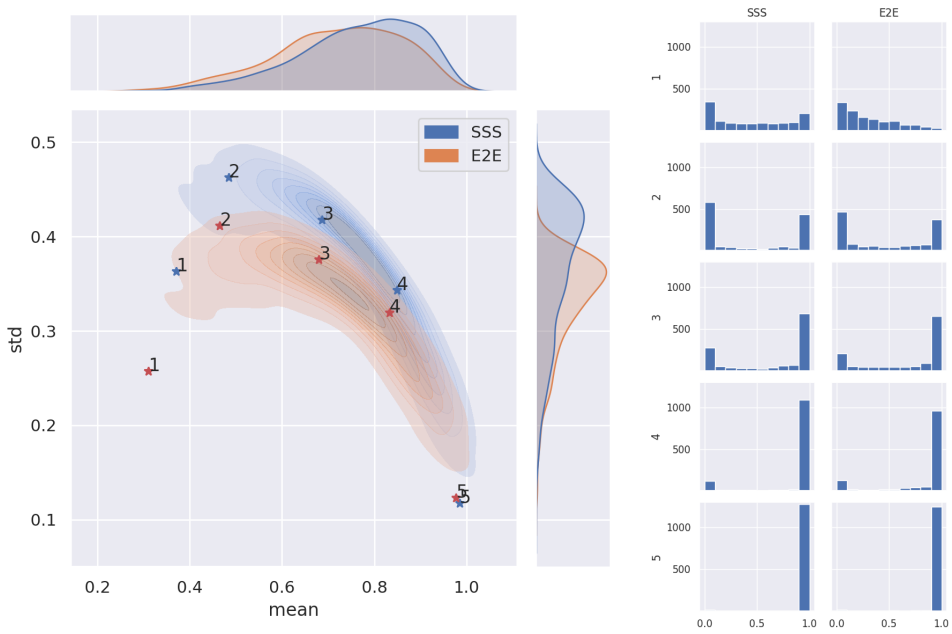


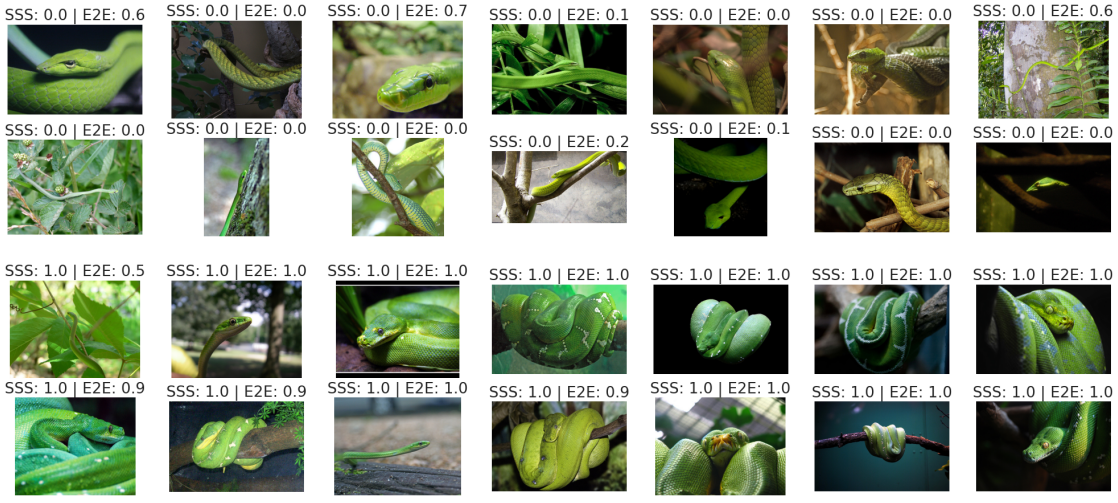
Figure 3.1 – SSS recovers the class-level stucture of E2E consistency. The highlighted classes correspond to Projectile (1), Weasel (2), Green Snake (3), Oscilloscope (4), Yellow Lady's Slipper (5). The figure on the right shows histograms of SSS and E2E consistency scores for each of these classes.

highly compute-efficient manner. We also plot histograms of consistency scores for these classes on the right of Figure 3.1, where we see broad similarities between the distributions under SSS and E2E; though notably, SSS scores tend to appear more bimodal as is especially clear in the scores for Projectiles.

We further visualize the consistency scores in Figure 3.2; for each of the previously identified five classes, we display a random selection of the most inconsistent (first two rows, consistency score 0) and consistent (third and fourth rows, consistency score 1) images. These results remain human interpretable, with inconsistent samples identified by SSS being clearly more atypical or mislabelled. For example, the first image in the second row of projectiles displays an airplane dropping bombs, the third image in the first row of weasels shows a woman pulling a weasel away from a man's neck, and the sixth image in the first row of oscilloscopes has a man as the central figure in the image (with the oscilloscope in the background).

(a) Projectiles

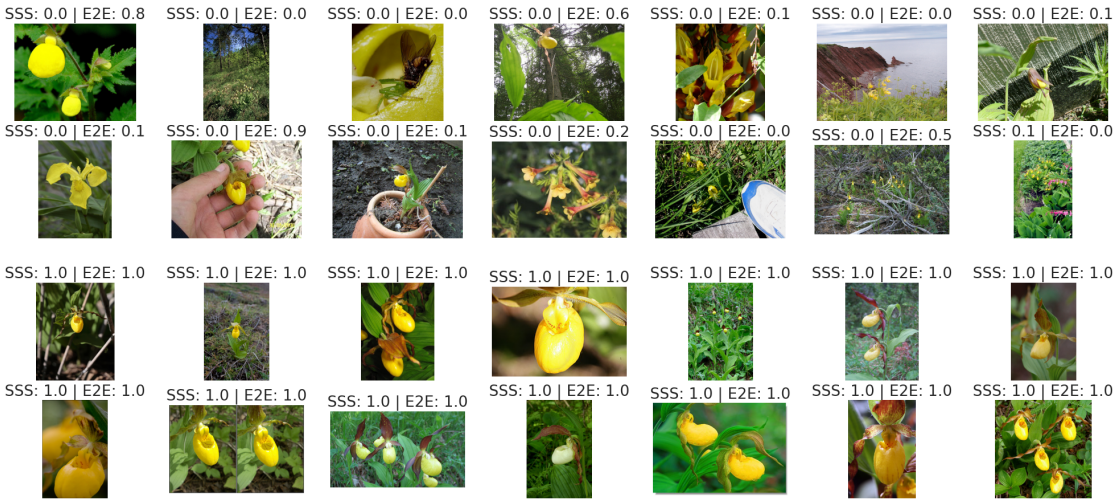
(b) Weasels



(c) Green Snakes

Figure 3.2 – Score histograms and inconsistent and consistent images, for classes identified as characteristic of ImageNet by (Jiang et al., 2020).

(d) Oscilloscopes



(e) Yellow Lady's Slippers

Figure 3.2 – (Continued) Score histograms and inconsistent and consistent images, for classes identified as characteristic of ImageNet by (Jiang et al., 2020).

Table 3.3 – Estimator resource usage in GPU-hours

Method	pretrain	Train
E2E-RN50	–	800,000
SWaV-RN50	16,000	40
CLIP-ViT	$< 73{,}728$	40

3.3.3 Resource Usage

Measuring memorization and influence in Feldman and Zhang (2020) and consistency in Jiang et al. (2020) on Imagenet with ResNet50s required 800,000 P100 GPU-hours – 8 GPUs were used for each of 2000 runs, and each run took 50 hours. In contrast, under the SSS approach, each of the 2000 linear classifiers takes 10 minutes to train, and 8 can be trained in parallel on a single GPU; in total we only require 40 RTX8000 GPU-hours, a 20,000x speedup compared to the E2E approach. We note that this computational requirement is even less than the cost of training a single ResNet50 on ImageNet-1K.

Of course, the vastly reduced computational requirements depended on the availability of strong pretrained visual encoders; the pretrained SwAV encoder required about 16,000 GPU-hours to train, and CLIP required on the order of 73,728 GPU-hours[3] to train. However, much as NLP has moved to a paradigm in which large self-supervised pretrained models are readily available and almost always used to finetune on specific tasks, we believe that a similar paradigm is taking hold in computer vision, and that these models can be taken for granted without extra compute requirements.

3.4 Revisiting the Long-Tail Theory

A motivation of Feldman and Zhang (2020) in developing the LMI-estimator was to provide empirical support for the Long-Tail Theory of Feldman (2019). This theory proposes that memorization of training data is beneficial or even necessary for optimal performance when data is drawn from long-tailed distributions (as is common in many natural settings) — in order to classify a rare or atypical test sample, memorizing a similar training sample might be the only solution. We conclude this work by briefly evaluating these claims under the SSS paradigm.

3. This is actually an upper-bound – CLIP only reported compute usage for their largest model, yet we are using their smallest model which was the only one publicly released.

Table 3.4 – High-influence pairs identified under various models and datasets. Test samples can be influenced my multiple training samples, so we report both the number of pairs and unique test samples.

Test Dataset	Model	Pairs	Test Samples (%)	Dim.	Acc.
ImageNet	E2E-RN50	1641	1462 (2.9%)	—	73%
ImageNet	SwAV-RN50	1233	933 (1.9%)	2048	73%
ImageNet	CLIP-ViT	596	514 (1.0%)	512	75%
ImageNetV2	CLIP-ViT	91	80 (0.8%)	512	63%
CIFAR-100	E2E-RN50	1015	888 (8.9%)	—	76%
CIFAR-100	CLIP-ViT	206	177 (1.8%)	512	
ciFAIR-100	CLIP-ViT	211	173 (1.7%)	512	

Memorization remains useful under SSS. The empirical findings of Feldman and Zhang (2020) indicated that correct classification of 2.92% of the ImageNet test-set and 8.88% of the CIFAR-100 test set was highly influenced by the presence of memorized samples in the corresponding training sets. They defined a high-influence train-test pair to be one where the memorization score of the training sample is > 0.25, and the influence score of that training sample on the test sample is > 0.15. In the SSS setting, in spite of having significantly reduced capacity to memorize labels, we see in Table 3.4 that on ImageNet roughly 1.9% of the test set benefitted from memorization under the SwAV encoder, and 1% under the CLIP encoder.

Memorization remains useful under distribution-shift or de-duplication with SSS. A visual inspection of E2E high-influence pairs revealed that many of them were near-duplicates between the train and test samples; differentiated by camera angles or temporal delay. This is an idiosyncracy of the ImageNet, as well as CIFAR100, dataset collection process– finding such near-duplicates would be exceedingly unlikely when sampling from the natural distribution of images which these datasets aim to emulate. We assess to what extent this affects the importance of memorization to test set classification, by replicating the high-influence calculations on new test datasets: ciFAIR (Barz and Denzler, 2020), which removed all (near-)duplicates of training images from the test set of CIFAR-100 and replaced them

with new images, and ImageNetV2 Recht et al. (2019), which replicated the data collection and labelling strategy of ImageNet yet suffers from distribution shift that causes ImageNet-trained models to perform significantly worse on it. Somewhat surprisingly, we find that memorization appears to remain beneficial under these settings; the proportion of influenced test samples on ImageNet drops from 1% to 0.8%, and the proportion on CIFAR-100 negligbly drops from 1.77% to 1.73%.

We speculate that while SSS and E2E reach similar test accuracies, the set of test points they correctly classify are subtly different. In particular, it is possible that test samples which benefitted from E2E memorization are not classified at all by SSS, as SSS may not have memorized the relevant training samples. This would help to explain why adapting to new test sets saw minimal reduction in the benefit of memorization. We intend to more closely evaluate the differences in test set predictions between SSS and E2E in future work.

4 Pretraining Representations for Data-Efficient Reinforcement Learning

Authors Max Schwarzer[1,2], Nitarshan Rajkumar[1,2], Michael Noukhovitch[1,2], Ankesh Anand[1,2], Laurent Charlin[1,3], Devon Hjelm[1,4], Phillip Bachman[4], Aaron Courville[1,2]

Affiliations [1]Mila, [2]Université de Montréal, [3]HEC Montréal, [4]Microsoft Research

Abstract Data efficiency is a key challenge for deep reinforcement learning. We address this problem by using unlabeled data to pretrain an encoder which is then finetuned on a small amount of task-specific data. To encourage learning representations which capture diverse aspects of the underlying MDP, we employ a combination of latent dynamics modelling and unsupervised goal-conditioned RL. When limited to 100k steps of interaction on Atari games (equivalent to two hours of human experience), our approach significantly surpasses prior work combining offline representation pretraining with task-specific finetuning, and compares favourably with other pretraining methods that require orders of magnitude more data. Our approach shows particular promise when combined with larger models as well as more diverse, task-aligned observational data – approaching human-level performance and data-efficiency on Atari in our best setting. We provide code associated with this work at https://github.com/mila-iqia/SGI.

Contributions I came up with the idea of applying SPR to pretrain representations on offline data, scaling this approach to larger models, and transferring representations between tasks (games). Max and I jointly designed the approach and experiments. I conducted initial experimentation to demonstrate the feasibility of this approach, and then focused on the experiments for demonstrating transfer across games (which were ultimately unsuccessful). Max formulated the goal-conditioned objective, and was responsible for all non-transfer experiments presented in the paper. I contributed significantly to the structure and writing of the paper itself, but this writing was led and largely done by Max. Michael as-

sisted with experiments for transfer, and writing of the paper. Ankesh contributed to feedback on the project and editing of the paper, as well as creating Figure 4.1. Laurent, Devon, Phil, and Aaron all contributed to supervision of the project and editing of the paper.

Submission An earlier version of this project was presented at the Self-Supervision for Reinforcement Learning Workshop at ICLR 2021. The work as presented here, verbatim, is under review for the conference track of NeurIPS 2021, and is available on arXiv (Schwarzer et al., 2021b).

4.1 Introduction

Deep reinforcement learning (RL) methods often focus on training networks *tabula rasa* from random initializations without using any prior knowledge about the environment. In contrast, humans rely a great deal on visual and dynamics priors about the world to perform decision making efficiently (Dubey et al., 2018; Lake et al., 2017). Thus, it is not surprising that RL algorithms which learn *tabula rasa* suffer from severe overfitting (Zhang et al., 2018) and poor sample efficiency compared to humans (Tsividis et al., 2017).

Self-supervised learning (SSL) provides a promising approach to learning useful priors from past data or experiences. SSL methods leverage unlabelled data to learn strong representations, which can be used to bootstrap learning on downstream tasks. Pretraining with self-supervised learning has been shown to be quite successful in vision (Hénaff et al., 2019; Grill et al., 2020a; Chen et al., 2020b) and language (Devlin et al., 2019; Brown et al., 2020b) settings.

Pretraining can also be used in an RL context to learn priors over representations or dynamics. One approach to pretraining for RL is to train agents to explore their environment in an unsupervised fashion, forcing them to learn useful skills and representations (Hansen et al., 2020; Liu and Abbeel, 2021; Campos et al., 2021). Unfortunately, current unsupervised exploration methods require months or years of real-time experience, which may be impractical for real-world systems with limits and costs to interaction — agents cannot be run faster than real-time, may require significant human oversight for safety, and can be expensive to run in parallel. It is

thus important to develop pretraining methods that work with practical quantities of data, and ideally that can be applied *offline* to fixed datasets collected from prior experiments or expert demonstrations (as in Stooke et al., 2021).

To this end, we propose to use a combination of self-supervised objectives for representation learning on offline data, requiring orders of magnitude less pretraining data than existing methods, while approaching human-level data-efficiency when finetuned on downstream tasks. We summarize our work below:

RL-aligned representation learning objectives: We propose to pretrain representations using a combination of latent dynamics modeling, unsupervised goal-conditioned reinforcement learning, and inverse dynamics modeling – with the intuition that a collection of such objectives can capture more information about the dynamical and temporal aspects of the environment of interest than any single objective. We refer to our method as **SGI** (**S**PR, **G**oal-conditioned RL and **I**nverse modeling).

Significant advances for data-efficiency on Atari: SGI's combination of objectives performs better than any in isolation and significantly improves performance over previous representation pretraining baselines such as ATC (Stooke et al., 2021). Our results are competitive with exploration-based approaches such as APT or VISR (Liu and Abbeel, 2021; Hansen et al., 2020), which require two to three orders of magnitude more pretraining data and the ability to interact with the environment during training, while SGI can learn with a small offline dataset of exploration data.

Scaling with data quality and model size: SGI's performance also scales with data quality and quantity, increasing as data comes from better-performing or more-exploratory policies. Additionally, we find that SGI's performance scales robustly with model size; while larger models are unstable or bring limited benefits in standard RL, SGI pretraining allows their finetuning performance to significantly exceed that of smaller networks.

We assume familiarity with RL in the following sections (with a brief overview in Section 1.4).

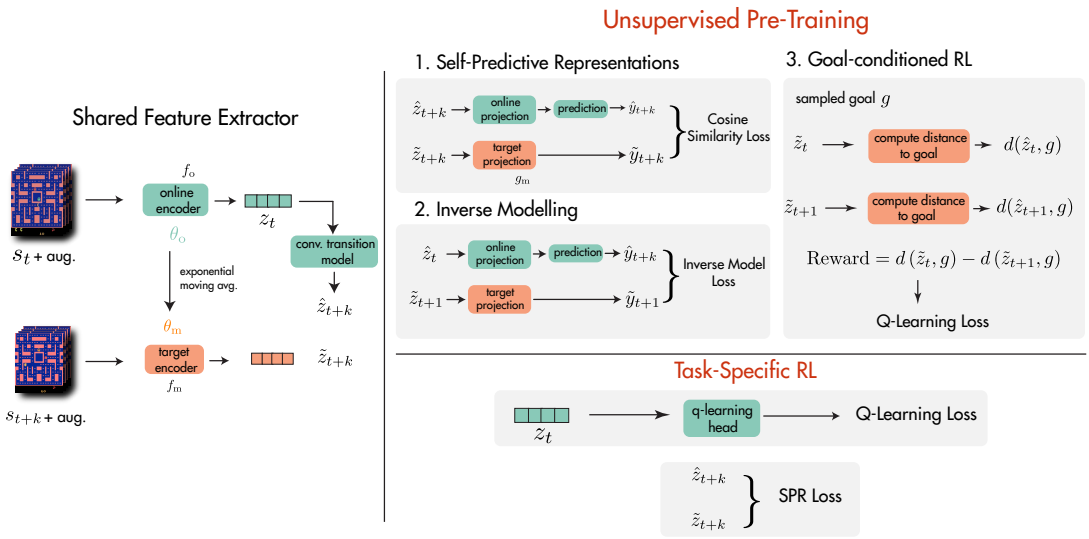


Figure 4.1 – A schematic diagram showing our two stage pretrain-then-finetune method. All unsupervised training losses and task-specific RL use the shared torso on the left.

4.2 Representation Learning Objectives

A wide range of SSL objectives have been proposed for RL which leverage various aspects of the structure available in agent interactions. For example, the temporal dynamics of an environment can be exploited to create a forward prediction task (e.g., Gelada et al., 2019; Guo et al., 2018; Stooke et al., 2021; Schwarzer et al., 2021a) in which an agent is trained to predict its immediate future observations, perhaps conditioned on a sequence of actions to perform.

Structure in RL goes far beyond forward dynamics, however. We propose to combine multiple representation learning objectives, covering different agent-centric and temporal aspects of the MDP. Based on the intuition that knowledge of an environment is best represented in multiple ways (Hessel et al., 2021; Degris and Modayil, 2012), we expect this to outperform monolithic representation learning methods such as temporal contrastive learning (e.g., Stooke et al., 2021). In deciding which tasks to use, we consider questions an adequate representation should be able to answer about its environment, including:

— If I take action a in state s, what state s' am I likely to see next?

— If I transitioned from state s to state s', what action a did I take?

— What action a would I take in state s so that I reach another state s' as soon as possible

Note that none of these questions are tied to task reward, allowing them to be answered in a fully-unsupervised fashion. Additionally, these are questions about the environment, and not any specific policy, allowing them to be used in offline pretraining with arbitrary behavioral policies.

In general, the first question may be answered by forward dynamics modeling, which as mentioned above is well-established in RL. The second question corresponds to inverse dynamics modeling, which has also been commonly used in the past (Lesort et al., 2018). The third question corresponds to self-supervised goal-conditioned reinforcement learning which has the advantage of being structurally similar to the downstream target task, as both require learning to control the environment.

To facilitate their joint use, we formulate these objectives so that they operate in the latent representation space provided by a shared encoder. We provide an overview of these components in Figure 4.1 and describe them in greater detail below; we also provide detailed pseudocode in Appendix A.3.

4.2.1 Self-Predictive Representations

SPR (Schwarzer et al., 2021a) is a representation learning algorithm developed for data-efficient reinforcement learning. SPR learns a latent-space transition model, directly predicting representations of future states without reconstruction or negative samples. As in its base algorithm, Rainbow (Hessel et al., 2018), SPR learns a convolutional encoder, denoted as f_o, which produces representations of states as $z_t = f_o(s_t)$. SPR then uses a *dynamics model* h to recursively estimate the representations of future states, as $\hat{z}_{t+k+1} = h(\hat{z}_{t+k}, a_{t+k})$, beginning from $\hat{z}_t \triangleq z_t$. These representations are projected to a lower-dimensional space by a projection function p_o to produce $\hat{y}_{t+k} \triangleq p_o(\hat{z}_{t+k})$.

Simultaneously, SPR uses a *target encoder* f_m to produce target representations $\tilde{z}_{t+k} \triangleq f_m(s_{t+k})$, which are further projected by a target projection function p_m to produce $\tilde{y}_{t+k} \triangleq p_m(\tilde{z}_{t+k})$. SPR then maximizes the cosine similarity between these predictions and targets, using a learned linear prediction function q to translate

from $\hat{y}$ to $\tilde{y}$:

$$\mathcal{L}_\theta^{\text{SPR}}(s_{t:t+K}, a_{t:t+K}) = -\sum_{k=1}^{K} \frac{q(\hat{y}_{t+k}) \cdot \tilde{y}_{t+k}}{||q(\hat{y}_{t+k})||_2 \cdot ||\tilde{y}_{t+k}||_2}. \tag{4.1}$$

The parameters of these target modules θ_m are defined as an exponential moving average of the parameters θ_o of f_o and p_o: $\theta_m = \tau\theta_m + (1-\tau)\theta_o$.

4.2.2 Goal-Conditioned Reinforcement Learning

Inspired by works such as Dabney et al. (2021) that show that modeling many different value functions is a useful representation learning objective, we propose to augment SPR with an unsupervised goal-conditioned reinforcement learning objective. We define goals g to be normalized vectors of the same size as the output of the agent's convolutional encoder (3,136- or 4,704-dimensional vectors, for the architectures we consider). We use these goals to annotate transitions with synthetic rewards, and train a modified version of Rainbow (Hessel et al., 2018) to estimate $Q(s_t, a, g)$, the expected return from taking action a in state s_t to reach goal g if optimal actions are taken in subsequent states.

We select goals using a scheme inspired by hindsight experience replay (Andrychowicz et al., 2017), seeking to generate goal vectors that are both semantically meaningful and highly diverse. As in hindsight experience replay, we begin by sampling a state from another trajectory or the future of the current trajectory. However, we take the additional step of applying stochastic noise to encourage goals to lie somewhat off of the current representation manifold. We provide details in Appendix A.2.2.

4.2.3 Inverse Dynamics Modeling

We propose to use an inverse dynamics modeling task (Lesort et al., 2018), in which the model is trained to predict a_t from s_t and s_{t+1}. Because this is a classification task (in discrete control) or regression task (continuous control), it is naturally not prone to representational collapse, which may complement and stabilize our other objectives. We directly integrate inverse modeling into the rollout structure of SPR, modeling $p(a_{t+k}|\hat{y}_{t+k}, \tilde{y}_{t+k+1})$ for each $k \in (0, \ldots, K-1)$,

using a two-layer MLP trained by cross-entropy.

4.3 Related Work

Data-Efficiency In order to address data efficiency in RL, Kaiser et al. (2019) introduced the Atari 100k benchmark, in which agents are limited to 100,000 steps of environment interaction, and proposed SimPLe, a model-based algorithm that substantially outperformed previous model-free methods. However, van Hasselt et al. (2019) and Kielak (2020) found that simply modifying the hyperparameters of existing model-free algorithms allowed them to exceed SimPLe's performance. Later, DrQ (Kostrikov et al., 2021) found that adding mild image augmentation to model-free methods dramatically enhanced their sample efficiency, while SPR (Schwarzer et al., 2021a) combined data augmentation with an auxiliary self-supervised learning objective. SGI employs SPR as one of its objectives in offline pretraining, leading to significant improvements in data-efficiency.

Exploratory pretraining A number of recent works have sought to improve reinforcement learning via the addition of an unsupervised *pretraining* stage prior to finetuning on the target task. One common approach has been to allow the agent a period of fully-unsupervised interaction with the environment, during which the agent is trained to maximize a surrogate exploration-based task such as the diversity of the states it encounters, as in APT (Liu and Abbeel, 2021), or to learn a set of skills associated with different paths through the environment, as in DIAYN (Eysenbach et al., 2018), VISR (Hansen et al., 2020), and DADS (Sharma et al., 2019). Others have proposed to use self-supervised objectives to generate intrinsic rewards encouraging agents to visit new states; e.g. Pathak et al. (2017) and Burda et al. (2018) use the loss of an inverse dynamics model like that used in SGI, while Sekar et al. (2020) uses the disagreement between an ensemble of latent-space dynamics models. Finally, Campos et al. (2021) report strong results based on massive-scale unsupervised pretraining.

Many of these methods are used to pretrain agents that are later adapted to specific reinforcement learning tasks. However, SGI differs in that it can be used offline and is agnostic to how data is collected. As such, if no pre-existing offline

data is available, one of the methods above can be used to generate a dataset for SGI; we use Burda et al. (2018) for this in Section 4.4.2.

Visual Representation Learning Computer vision has seen a series of dramatic advances in self-supervised representation learning, including contrastive methods (Oord et al., 2018; Hjelm et al., 2019; Bachman et al., 2019; He et al., 2020; Chen et al., 2020b) as well as purely predictive ones (Grill et al., 2020a). Variants of these approaches have also been shown to improve performance when coupled with a small quantity of labeled data, in a *semi-supervised* setting (Chen et al., 2020c; Hénaff et al., 2019), and several self-supervised methods have been designed specifically for this case (for example, Sohn et al., 2020; Tarvainen and Valpola, 2017).

These advances have spurred similar growth in methods aimed specifically at improving performance in RL. We refer the reader to Lesort et al. (2018) for a review of earlier methods, including inverse dynamics modeling which is used in SGI. Recent research has focused on leveraging latent-space dynamics modeling as an auxiliary task. Gelada et al. (2019) propose a simple next-step prediction task, coupled with reward prediction, but found it is prone to latent space collapse and requires an auxiliary reconstruction loss for experiments on Atari. Guo et al. (2020) use a pair of networks for both forward and backward prediction, and show improved performance in extremely large-data multi-task settings. Mazoure et al. (2020) use a temporal contrastive objective for representation learning, and show improvement in continual RL settings. Concurrently, SPR (Schwarzer et al., 2021a) proposed a multi-step latent prediction task with similarities to BYOL (Grill et al., 2020a).

Two works similar to ours, Anand et al. (2019) and Stooke et al. (2021), propose reward-free temporal-contrastive methods to pretrain representations. Anand et al. (2019) show that representations from encoders trained with ST-DIM contain a great deal of information about environment states, but they do not examine whether or not representations learned via their method are, in fact, useful for reinforcement learning. However, Stooke et al. (2021) employ a similar algorithm and find only relatively minor improvements in performance compared to standard baselines in the large-data regime; our controlled comparisons show that SGI's representations are far better for data-efficiency. Concurrent to our work, FERM

(Zhan et al., 2020) propose contrastive pretraining from human demonstrations in a robotics setting. As FERM is quite similar to ATC, we are optimistic that our improvements over ATC in Atari 100k would translate to FERM's setting.

4.4 Experimental Details

In our experiments, We seek to address two main challenges for the deployment of RL agents in the real world (Dulac-Arnold et al., 2020): (1) training the RL agent with a limited budget of interactions in the real environment, and (2) leveraging existing interaction data of **arbitrary quality**.

4.4.1 Environment and Evaluation

To address the first challenge, we focus our experimentation on the Atari 100k benchmark introduced by Kaiser et al. (2019), in which agents are allowed only 100k steps of interaction with their environment.[1] This is roughly equivalent to the two hours human testers were given to learn these games by Mnih et al. (2015), providing a baseline of human sample-efficiency.

Atari is also an ideal setting due to its complex observational spaces and diverse tasks, with 26 different games included in the Atari 100k benchmark. These factors have led to Atari's extensive use for representation learning and exploratory pretraining (Anand et al., 2019; Stooke et al., 2021; Campos et al., 2021), and specifically Atari 100k for data-efficient RL (e.g., Kaiser et al., 2019; Kostrikov et al., 2021; Schwarzer et al., 2021a), including finetuning after exploratory pretraining (e.g., Hansen et al., 2020; Liu and Abbeel, 2021), providing strong baselines to compare to.

Our evaluation metric for an agent on a game is *human-normalized score* (HNS), defined as $\frac{agent\_score - random\_score}{human\_score - random\_score}$. We calculate this per game by averaging scores over 100 evaluation trajectories at the end of training, and across 10 random seeds for training. We report both mean (Mn) and median (Mdn) HNS over the 26 Atari-100K games, as well as on how many games a method achieves super-human performance (>H) and greater than random performance (>0). Following the

1. Note that sticky actions are disabled under this benchmark.

guidelines of Anonymous (2021) we also report interquartile mean HNS (IQM) and quantify uncertainty via bootstrapping in Appendix A.1.

4.4.2 Pretraining Data

The second challenge pertains to pretraining data. Although some prior work on offline representational pretraining has focused on expert-quality data (Stooke et al., 2021), we expect real-world pretraining data to be of greatly varying quality. We thus construct four different pretraining datasets to approximate different data quality scenarios.

— **(R)andom** To assess performance near the lower limit of data quality, we use a random policy to gather a dataset of 6M transitions for each game. To encourage the agent to venture further from the starting state, we execute each action for a random number of steps sampled from a Geometric($\frac{1}{3}$) distribution.

— **(E)xploratory** To emulate slightly better data that covers a larger range of the state space, we use an exploratory policy. Specifically, we employ the **IDF** (inverse dynamics) variant of the algorithm proposed by Burda et al. (2018). We log the first 6M steps from an agent trained in each game. This algorithm achieves better-than-random performance on only 70% of tasks, setting it far below the performance of more modern unsupervised exploration methods.

To create higher-quality datasets, we follow Stooke et al. (2021) and use experience gathered during the training of standard DQN agents (Mnih et al., 2015). We opt to use the publicly-available DQN Replay dataset (Agarwal et al., 2020), which contains data from training for 50M steps (across all 57 games, with five different random seeds). Although we might prefer to use data from recent unsupervised exploration methods such as APT (Liu and Abbeel, 2021), VISR (Hansen et al., 2020), or CPT (Campos et al., 2021), none of these works provide code or datasets, making this impractical. We address using data collected from on-task agents with a behavioural cloning baseline in Section 4.5, with surprising findings relative to prior work.

— **(W)eak** We first generate a weak dataset by selecting the first 1M steps for each of the 5 available runs in the DQN Replay dataset. This data is generated with an ϵ-greedy policy with high, gradually decaying ϵ, leading to substantial action

diversity and many suboptimal exploratory actions. Although the behavioral policies used to generate this agent are not especially competent (see Table 4.1), they have above-random performance on almost all games, suggesting that that this dataset includes more task-relevant transitions.

— **(M)ixed** Finally, for a realistic best-case scenario, we create a dataset of both medium and low-quality data. To simulate a real-world collection of data from different policies, we concatenate multiple checkpoints evenly spread throughout training of a DQN. We believe this is also a reasonable approximation for data from a modern unsupervised exploration method such as CPT (Campos et al., 2021); as shown in Table 4.1, the agent for this dataset has performance in between CPT and VISR, with median closer to CPT and mean closer to VISR. This data is also lower quality than the expert data originally used in the method most similar to ours, ATC (Stooke et al., 2021).[2] We create a dataset of 3M steps and a larger dataset of 6M steps; all **M** experiments use the 3M step dataset unless otherwise noted.

Table 4.1 – Performance of agents used in pretraining data collection compared to external baselines on 26 Atari games (Kaiser et al., 2019)

Method	Mdn	Mean	>H	>0	Data
Exploratory Pretraining Baselines					
VISR@0	0.056	0.817	5	19	250M
APT@0[1]	0.038	0.476	2	18	250M
CPT@0	**0.809**	**4.945**	**12**	25	4B
Offline Datasets					
Exploratory	0.039	0.042	0	18	6M
Weak[2]	0.028	0.056	0	23	5M
Mixed[2]	0.618	1.266	10	**26**	3M

[1] Calculated from ICLR 2021 OpenReview submission; unreported in arXiv version.
[2] Upper-bound estimate from averaging evaluation performance of corresponding agents in Dopamine.

2. Our data-collection agents are weaker than those used by ATC on seven of the eight games they consider.

We compare the agents used for our datasets to those for unsupervised exploration pretraining baselines in Table 4.1. We estimate the performance of the Weak and Mixed agents as the average of the corresponding logged evaluations in the Dopamine (Castro et al., 2018) baselines. Even our largest dataset is quite small compared to the amounts of data gathered by unsupervised exploration methods (see the "Data" column in Table 4.1); this is intentional, as we believe that unsupervised interactional data may be expensive in real world applications. We show the performance of the non-random data collection policies in Table 4.2 (note that a fully-random policy has a score of **0** by definition).

4.4.3 Training Details

We optimize our three representation learning objectives jointly during unsupervised pretraining, summing their losses. During finetuning, we optimize only the reinforcement learning and forward dynamics losses, following Schwarzer et al. (2021a) (see Section 4.5.5), and lower the learning rates for the pretrained encoder and dynamics model by two orders of magnitude (see Section 4.5.4).

We consider the standard three-layer convolutional encoder introduced by Mnih et al. (2015), a ResNet inspired by Espeholt et al. (2018), as well as an enlarged ResNet of the same design. In other respects, our implementation matches that of SPR and is based on its publicly-released code. Full implementation and hyperparameter details are provided in Appendix A.2. We refer to agents by the model architecture and pretraining dataset type used: **SGI-R** is pretrained on Random, **SGI-E** on Exploratory, **SGI-W** on Weak, and **SGI-M** on Mixed. To investigate scaling, we vary the size of the encoder used in **SGI-M**: the standard Mnih et al. (2015) encoder is **SGI-M/S** (for small), our standard ResNet is simply **SGI-M** and using a larger ResNet is **SGI-M/L** (for large)[3]. For **SGI-M/L** we also use the 6M step dataset described earlier. All ablations are conducted in comparison to **SGI-M** unless otherwise noted. Finally, agents without pretraining are denoted **SGI-None**; SGI-None/S would be roughly equivalent to SPR (Schwarzer et al., 2021a).

For baselines, we compare to no-pretraining Atari 100k methods (Kaiser et al., 2019; van Hasselt et al., 2019; Kostrikov et al., 2021; Schwarzer et al., 2021a). For our models trained on Random and Exploratory data we compare against

3. See Appendix A.2 for details on these networks

previous pretraining-via-exploration approaches applied to Atari 100k (Liu and Abbeel, 2021; Hansen et al., 2020; Campos et al., 2021). In the higher quality data regime, we compare to recent work on data-agnostic unsupervised pretraining, ATC (Stooke et al., 2021), as well as behavioural cloning (BC).

4.5 Results and Discussion

We find that SGI performs competitively on the Atari-100K benchmark; presenting aggregate results in Table 4.2, and full per-game data in Appendix A.4. Our best setting, **SGI-M/L**, achieves a median HNS of 0.753, approaching human-level sample-efficiency and outperforming all comparable methods except the recently proposed CPT (Campos et al., 2021). With less data and a smaller model, **SGI-M** achieves a median HNS of 0.679, significantly outperforming the prior method ATC on the same data (**ATC-M**). Meanwhile, **SGI-E** achieves a median HNS of 0.456, matching or exceeding other exploratory methods such as APT (Liu and Abbeel, 2021) and VISR (Hansen et al., 2020), as well as ATC-E.

Pretraining data efficiency SGI achieves strong performance with only limited pretraining data; our largest dataset contains 6M transitions, or roughly 4.5 days of experience. This compares favourably to recent works on unsupervised exploration such as APT or CPT, which require far larger amounts of data and environment interaction (250M steps or 193 days for APT, 4B steps or 8.45 years for CPT). We expect SGI would perform even better if used in these large-data settingss, as we find that it scales robustly with data (see Section 4.5.2).

Behavioural cloning is a strong baseline Although ATC pretrains with expert data, they did not investigate behavioral cloning as a baseline pretraining objective. We do so on our **Mixed** dataset, the only one to be generated by policies with significantly above-random performance. Behavioral cloning without finetuning (**BC-M@0**) performs poorly, perhaps due to the varying behavioural quality in the dataset. But when finetuned, **BC-M** yields very respectable performance, surpassing **ATC-M** but not **SGI-M**. All fine-tuning settings for BC-M match SGI-M.

Table 4.2 – HNS on Atari100k for SGI and baselines.

Method	Mdn	Mn	>H	>0	Data
No Pretraining (Finetuning Only)					
SimPLe	0.144	0.443	2	**26**	0
DER	0.161	0.285	2	**26**	0
DrQ	0.268	0.357	2	24	0
SPR	0.415	0.704	7	**26**	0
SGI-None	0.343	0.565	3	26	0
Exploratory Pretraining + Finetuning					
Method	Mdn	Mn	>H	>0	Data
VISR	0.095	1.281	7	21	250M
APT	0.475	0.666[1]	7	**26**	250M
CPT@0[2]	**0.809**	**4.945**	12	25	4000M
ATC-R[3]	0.191	0.472	4	**26**	6M
ATC-E[3]	0.237	0.462	3	**26**	6M
SGI-R	0.326	0.888	5	**26**	6M
SGI-E	0.456	0.838	6	**26**	6M
Offline-data Pretraining + Finetuning					
Method	Mdn	Mn	>H	>0	Data
ATC-W[3]	0.219	0.587	4	**26**	3M
ATC-M[3]	0.204	0.780	5	**26**	3M
BC-M@0	0.139	0.227	0	23	3M
BC-M	0.548	0.858	8	**26**	3M
SGI-W	0.589	1.144	8	**26**	5M
SGI-M/S	0.423	0.914	8	**26**	3M
SGI-M	0.679	1.149	**9**	**26**	3M
SGI-M/L	**0.753**	**1.598**	**9**	**26**	6M

[1] APT claims 0.6955, but we calculate 0.666 from their reported per-game scores.
[2] CPT@0 does not do any finetuning.
[3] Our implementation (see Appendix A.2.5)

4.5.1 Data quality matters

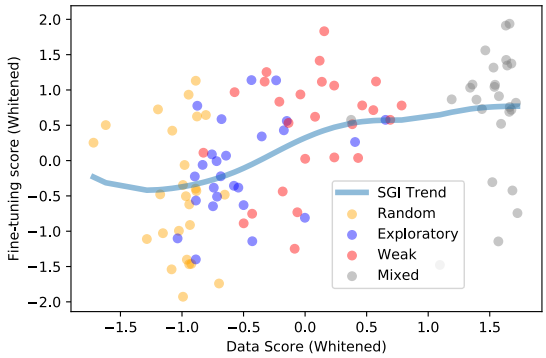


Figure 4.2 – SGI finetuning performance vs. pretraining data score for all combinations of game and dataset. Data score is estimated as clipped return per episode, trend calculated via kernel regression. Values whitened per-game for clarity.

In principle, SGI can be used with any offline dataset but we demonstrate that it scales with the quality of its data. Near the lower bound of data quality where all actions are selected randomly, **SGI-R** still provides some benefit over an otherwise-identical randomly-initialized agent (**SGI-None**) on 16 out of 26 games, with a similar median but 57% higher mean HNS. With better data from an exploratory policy, **SGI-E** improves on 16/26 games, gets 33% higher median HNS, and surpasses APT (Liu and Abbeel, 2021) which used 40 times more pretraining data. With similarly weak data but possibly more task-specific transitions, **SGI-W** gets 72% higher median HNS compared to SGI-None and with realistic data from a mixture of policies, **SGI-M** improves to 98%.

Importantly, the pattern we observe is very different from what would be expected for imitation learning. In particular, SGI-W's strong performance shows that expert data is not required. To characterize this, we plot the average clipped reward [4] experienced per episode for each of our pretraining datasets in Figure 4.2. Normalizing across tasks, we find a strong positive correlation between task reward engagement ($p < 0.0001$) and finetuning performance. Moreover, we find diminishing returns to further task engagement.

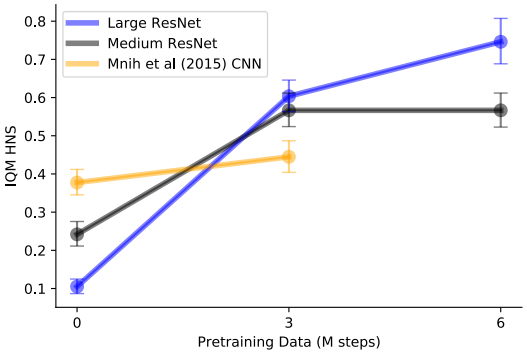


Figure 4.3 – Finetuning performance of SGI for different CNN sizes and amounts of pretrained data from the **Mixed** dataset. We plot IQM HNS with confidence intervals (see Appendix A.1).

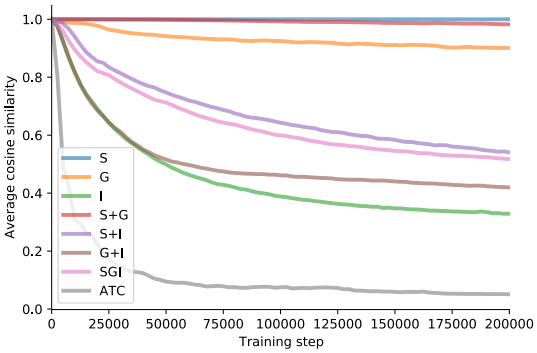


Figure 4.4 – Average cosine similarity between representations over pretraining, averaged across the 26 Atari 100k games. 1 indicates representations are identical, 0 perfect dissimilarity.

4.5.2 Pretraining unlocks the value of larger networks

The three-layer network introduced by Mnih et al. (2015) has become a fixture of deep reinforcement learning, and has been employed by previous works examining pretraining in this field (e.g. Liu and Abbeel, 2021; Stooke et al., 2021). However, we find that representational pretraining with this network (**SGI-M/S**) provides only minor benefits compared to training from scratch. In contrast, larger networks struggle without pretraining but shine when pretrained as shown in Figure 4.3.

This finding is consistent with recent work in unsupervised representation learning for classification, which has observed that unsupervised pretraining benefits disproportionately from larger networks (Chen et al., 2020b). In particular, our results suggest that model size should increase in parallel with amount of pretraining data, matching recent work on scaling in language modeling (Kaplan et al., 2020; Hernandez et al., 2021). SGI thus provides a simple way to use unlabeled data to extend the benefits of large networks, already well-known in the large-data regime (e.g., Schrittwieser et al., 2021; Espeholt et al., 2018), to data-efficient RL.

4.5.3 Combining SGI's objectives improves performance

We test all possible combinations of our three SSL objectives, denoted by combinations of the letters S, G and I to indicate which objectives they employ. Results in Table 4.3 show that performance monotonically increases as more objectives are

4. Unclipped rewards are not available for the offline DQN dataset.

Table 4.3 – HNS on Atari 100K for pretraining ablations of SGI.

Pretraining	Mdn	Mean	>H
None	0.343	0.565	3
S	0.009	-0.054	0
G	0.060	0.181	1
I	0.411	0.943	7
S+G	0.029	0.098	0
G+I	0.512	1.004	**9**
S+I	0.629	0.978	8
SGI-M	**0.679**	**1.149**	**9**

used, with inverse dynamics modeling combined with either of the other objectives performing respectably well. This illustrates the importance of using multiple objectives to obtain representational coverage of the MDP.

We note that including inverse modeling appears to be critical, and hypothesize that this is related to representational collapse. To measure this, we plot the average cosine similarity between representations y_t of different states for several pretraining methods in Figure 4.4, using our ResNet encoder on the Mixed dataset. We observe that **S**, **G** and **S+G** all show some degree representational collapse, while configurations that include inverse dynamics modeling avoid representational collapse, as does ATC, whose contrastive loss implicitly optimizes for representational diversity (Wang and Isola, 2020). Intriguingly, we observe that increased representational diversity does not necessarily improve performance. For example, SGI outperforms **ATC**, **G+I** and **I** in finetuning but has less diverse pretraining representations. We also observe that adding SPR (**S**) consistently pulls representations towards collapse (compare **S+I** and **I**, **S+G** and **G**, and **SGI** and **G+I**); how this relates to performance is a question for future work.

4.5.4 Naively finetuning ruins pretrained representations

We find that properly finetuning pretrained representations is critical, as results in Table 4.4 show. Although allowing pretrained weights to change freely during finetuning is better than initializing from scratch (**Naive** vs **No Pretrain**), freezing the pretrained encoder (**Frozen**) leads to better performance than either. SGI's

Table 4.4 – Human-normalized score on Atari-100K for various controlled comparisons to SGI-C.

Method	Mdn	Mean	>H	> SPR
Naive	0.429	0.845	8	14
Frozen	0.499	0.971	8	15
No SPR	0.452	1.114	8	14
All Losses	0.397	1.011	8	17
SGI-C	**0.679**	**1.149**	**9**	**20**

approach of reducing finetuning learning rates for pretrained parameters leads to superior performance (**Reduced LRs**, equivalent to **SGI-M**).

We thus hypothesize that representations learned by SGI are being disrupted by gradients early in finetuning, in a phenomenon analogous to catastrophic forgetting (Zenke et al., 2017; Hadsell et al., 2020). As representations may not generalize between different value functions across training (Dabney et al., 2021), allowing the encoder to strongly adapt early in training could make it *worse* at modeling later value functions, compared to the neutral initialization from SGI. We also note that there is a long history in computer vision of employing specialized finetuning hyperparameters (Li et al., 2020; Chu et al., 2016) when transferring tasks.

4.5.5 Not all SSL objectives are beneficial during finetuning

Table 4.5 – HNS on Atari 100K for finetuning ablations of SGI.

Finetune SSL	Mdn	Mean	>H
Without SGI pretraining			
None	0.161	0.315	2
S Only	0.343	0.565	3
With SGI pretraining			
None	0.452	1.114	8
SGI	0.397	1.011	8
S Only	**0.679**	**1.149**	**9**

Although SGI uses **S** during finetuning, we experiment with a variant that optimizes only the standard DQN objective, roughly equivalent to using DrQ (Kostrikov

et al., 2021) with DQN hyperparameters set to match SGI. We find that finetuning with **S** has a large impact with or without pretraining (compare **None** and **S Only** entries in Table 4.5.). Although, SGI without **S** manages to achieve roughly the same mean human-normalized score as SGI with **S**, removing **S** harms performance on performance on 19 out of 26 games and reduces median normalized score by roughly 33%. We also find no benefit to using all of SGI's constituent objectives during finetuning (**All Losses** in Table 4.5) compared to using **S** alone, although the gap between them is not statistically significant for metrics other than median (see Figure A.1d).

5 Conclusion

Sample Importance We have demonstrated that explicit separation of (overparameterized) representation learning and (underparameterized) task-specific classifiers appears to provide a useful surrogate model-class to fully-supervised models for the purposes of measuring sample importance, while requiring orders of magnitude less compute. There are a few avenues for further advancing this work, which we hope to investigate soon. The first is to evaluate SSS as a surrogate model for other sample importance estimators such as TracIn (Pruthi et al., 2020) and Influence Functions (Koh and Liang, 2017). The second is to investigate how much of these measures can be recovered using non-parametric approaches on the representations alone, such as with density estimation, or nearest-neighbour classifiers.

Data-Efficient RL We presented SGI, a fully self-supervised (reward-free) approach to representation learning for reinforcement learning, which uses a combination of pretraining objectives to encourage the agent to learn multiple aspects of environment dynamics. We demonstrate that SGI enables significant improvements on the Atari 100k data-efficiency benchmark, especially in comparison to unsupervised exploration approaches which require orders of magnitude more pretraining data. Investigating the various components of SGI, we find that it scales robustly with higher-quality pretraining data and larger models, that all three of the self-supervised objectives contribute to the success of this approach, and that careful reduction of fine-tuning learning rates is critical to optimal performance. An initial goal for this work was to learn representations which could transfer *across* tasks, which we were unable to make work successfully; we believe this to be a very important goal to keep working towards, perhaps on more realistic benchmark tasks such as in robotics or self-driving. Furthermore, we are interested in using SGI itself as a sample-efficient exploratory method.

A Appendix for Pretraining Representations for Data-Efficient Reinforcement Learning

A.1 Uncertainty-aware comparisons

Concurrent work (Anonymous, 2021) has found that many prior comparisons in deep reinforcement learning are not robust and may be entirely incorrect, particularly in the Atari 100K setting. They demonstrate that these misleading comparisons are partially due to undesirable properties of the per-game median and mean normalized scores, the most commonly-used aggregate metrics, and propose using the inter-quartile mean (IQM) normalized score, calculated over *runs* rather than *tasks*. Moreover, they suggest providing percentile bootstrap confidence intervals to quantify uncertainty, to avoid misleading comparisons based on highly-variable point estimates.

As raw per-run data is required for this, which was not reported for prior work, we do so only for experiments conducted ourselves. In the interests of improving practices in the community moving forward, we also commit to making this data for our experiments available to other researchers in the future.

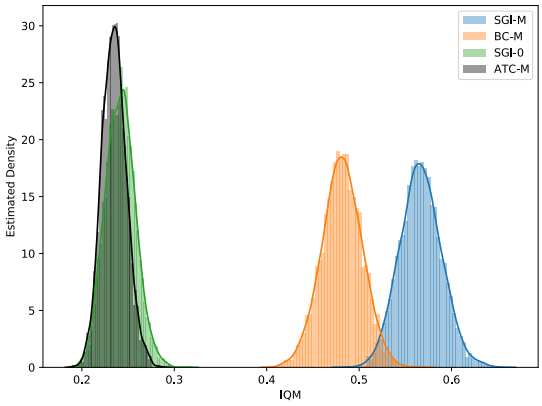
In Figure A.1a through Figure A.1e we show estimated uncertainty via bootstrapping for the various comparisons drawn throughout Section 4.5, while Table A.1 gives IQM human-normalized scores and 95% bootstrap confidence intervals for the same results. All comparisons in Figure A.1a through Figure A.1e are statistically significant ($p < 0.05$) except for:

— ATC-M vs SGI-None in Figure A.1a ($p \gg 0.05$)

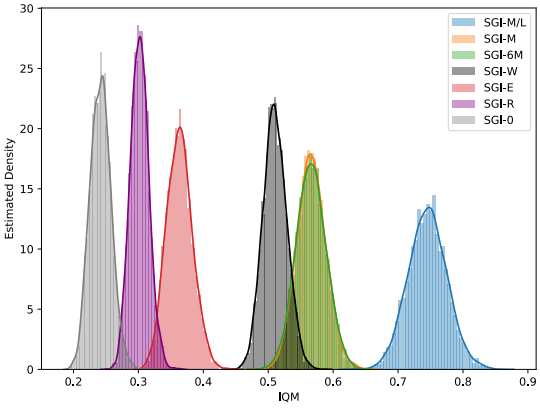
— SGI-M vs SGI-W in Figure A.1b ($p \approx 0.05$)

— SGI-M vs SGI-M w/ SGI FT in Figure A.1d ($p \approx 0.4$)

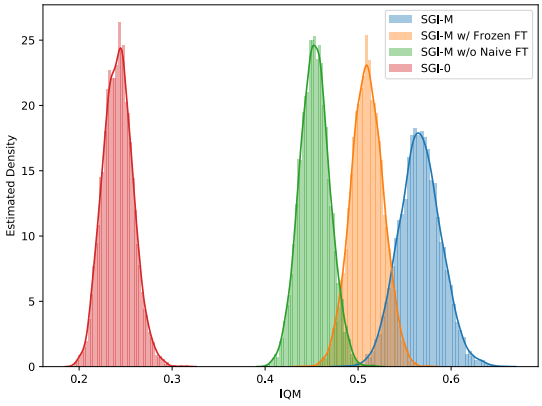
— SGI-M vs G+I and S+I in Figure A.1e ($p \approx 0.1$)



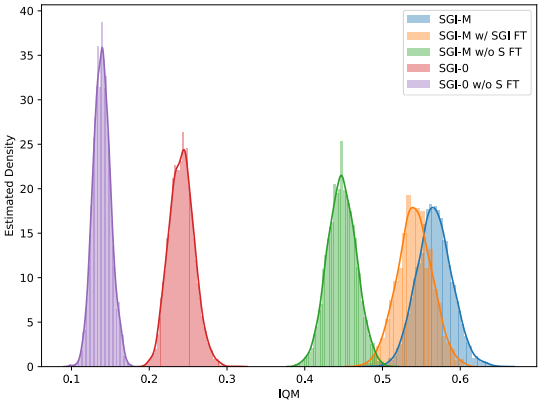
(a) Comparisons to behavioral cloning (BC) and ATC.



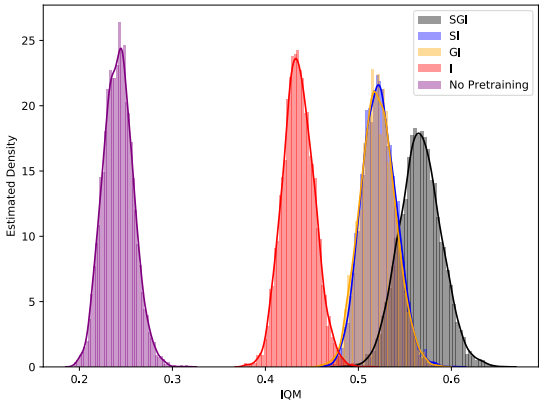
(b) Ablations over different pretraining datasets.



(c) Ablations over various fine-tuning.



(d) Ablations over SSL objectives during fine-tuning.



(e) Ablations over pretraining SSL objectives.

Figure A.1 – Bootstrapping distributions for uncertainty in IQM measurements.

Table A.1 – Interquartile mean, median and mean human-normalized scores for variants of SGI and controls, evaluated after finetuning over all 10 runs for each of the 26 Atari 100k games. Confidence intervals computed by percentile bootstrap with 5000 resamples.

Method	IQM	95% CI	Median	95% CI	Mean	95% CI
SGI-M/L	**0.745**	(0.687, 0.805)	**0.753**	(0.625, 0.850)	**1.598**	(1.486, 1.676)
SGI-M	0.567	(0.524, 0.612)	0.679	(0.473, 0.739)	1.149	(0.974, 1.347)
SGI-M/S	0.444	(0.404, 0.487)	0.423	(0.341, 0.577)	0.914	(0.822, 1.031)
SGI-W	0.510	(0.476, 0.547)	0.589	(0.434, 0.675)	1.144	(0.981, 1.345)
SGI-E	0.363	(0.326, 0.404)	0.456	(0.309, 0.482)	0.838	(0.692, 1.008)
SGI-R	0.302	(0.275, 0.331)	0.326	(0.253, 0.385)	0.888	(0.776, 1.004)
SGI-None	0.242	(0.212, 0.274)	0.343	(0.268, 0.401)	0.565	(0.440, 0.711)
Baselines						
ATC-M	0.235	(0.210, 0.262)	0.204	(0.182, 0.291)	0.780	(0.601, 0.971)
ATC-W	0.221	(0.199, 0.244)	0.219	(0.170, 0.290)	0.587	(0.504, 0.673)
ATC-E	0.214	(0.193, 0.236)	0.237	(0.169, 0.266)	0.462	(0.420, 0.504)
ATC-R	0.187	(0.174, 0.202)	0.191	(0.139, 0.202)	0.472	(0.454, 0.491)
BC-M	0.481	(0.438, 0.524)	0.548	(0.390, 0.685)	0.858	(0.795, 0.924)
Pretraining Ablations						
S+I	0.522	(0.488, 0.559)	0.629	(0.494, 0.664)	0.978	(0.900, 1.061)
G+I	0.521	(0.486, 0.558)	0.512	(0.386, 0.582)	1.004	(0.892, 1.129)
S+G	0.032	(0.027, 0.039)	0.029	(0.025, 0.044)	0.098	(0.061, 0.146)
I	0.435	(0.404, 0.470)	0.411	(0.334, 0.489)	0.943	(0.783, 1.126)
G	0.060	(0.048, 0.072)	0.060	(0.037, 0.081)	0.181	(0.145, 0.218)
S	0.007	(0.002, 0.011)	0.009	(0.002, 0.014)	-0.054	(-0.082, -0.026)
Finetuning Ablations						
SGI-M (No S)	0.448	(0.412, 0.484)	0.419	(0.335, 0.524)	1.114	(0.921, 1.321)
SGI-None (No S)	0.139	(0.118, 0.162)	0.161	(0.123, 0.225)	0.315	(0.274, 0.356)
SGI-M (All SGI)	0.541	(0.498, 0.585)	0.397	(0.330, 0.503)	1.011	(0.909, 1.071)
SGI-M (Frozen)	0.510	(0.476, 0.543)	0.499	(0.406, 0.554)	0.971	(0.871, 1.088)
SGI-M (Naive)	0.453	(0.422, 0.485)	0.429	(0.380, 0.500)	0.845	(0.754, 0.952)

A.2 Implementation Details

We base our work on the code released for SPR (Schwarzer et al., 2021a), which in turn is based on rlpyt (Stooke and Abbeel, 2019), and makes use of NumPy (Harris et al., 2020) and PyTorch (Paszke et al., 2019).

A.2.1 Training

We set $\lambda^{\text{SPR}} = 2$ and $\lambda^{\text{IM}} = 1$ during pre-training. Unless otherwise noted, all settings match SPR during fine-tuning, including batch size, replay ratio, target network update period, and λ^{SPR}. We use a batch size of 256 during pre-training to maximize throughput, and update both the SPR and goal-conditioned RL target network target networks with an exponential moving average with $\tau = 0.99$. We pre-train for a number of gradient steps equivalent to 10 epochs over 6M samples, no matter the amount of data used. Due to the cost of pretraining, we pre-train a single encoder per game for each configuration tested. However, we use 10 random seeds at fine-tuning time, allowing us to average over variance due to exploration and data order. Finally, we reduce fine-tuning learning rates for pretrained encoders and dynamics models by a factor of 100, and by a factor of 3 for other pretrained weights. We find this crucial to SGI's performance, and discuss it in detail in Section 4.5.4.

We trained SGI on standard GPUs, including V100s and P100s. We found that pretraining took roughly one to three days and finetuning between four and 12 hours per run on a single GPU, depending on the size of the network used and type of GPU.

A.2.2 Goal-Conditioned Reinforcement Learning

We generate goals in a three-stage process: a goal g for state s_t is initially chosen to be the target representation of a state sampled uniformly from the near future, $g \leftarrow \tilde{z}_{t+i}, i \sim \text{Uniform}(50)$, before being combined with a normalized vector of isotropic Gaussian noise n as $g \leftarrow \alpha n + (1-\alpha)g$, where $\alpha \sim \text{Uniform}(0, 0.5)$. Finally, we exchange goal vectors between states in the minibatch with probability 0.2, to ensure that some goals correspond to states reached in entirely different trajectories.

In defining our synthetic goal-conditioned rewards, we take inspiration from potential-based reward shaping (Ng et al., 1999). Using the target representations $\tilde{z}_t \triangleq f_m(s_t)$ and $\tilde{z}_{t+1} \triangleq f_m(s_{t+1})$, we define the reward as follows:

$$R(\tilde{z}_t, \tilde{z}_{t+1}, g) = d(\tilde{z}_t, g) - d(\tilde{z}_{t+1}, g) \tag{A.1}$$

$$d(\tilde{z}_t, g) = \exp\left(2\frac{\tilde{z}_t \cdot g}{||\tilde{z}||_2 \cdot ||g||_2} - 2\right). \tag{A.2}$$

As this reward function depends on the target encoder f_m, it changes throughout training, although using the slower-moving f_m rather than the online encoder f_o may provide some measure of stability. Like SPR, however, this objective is technically vulnerable to collapse. If all representations $\tilde{z}_t$ collapse to a single constant vector then all rewards will be 0, allowing the task to be trivially solved.

We estimate $Q(s_t, a_t, g)$ using FiLM (Perez et al., 2018) to condition the DQN on the goal g, which we found to be more robust than simple concatenation. A FiLM generator j produces per-channel biases β_c and scales γ_c, which then modulate features through a per-channel affine transformation:

$$\text{FiLM}(F_c|\gamma_c, \beta_c) = \gamma_c F_c + \beta_c \tag{A.3}$$

We use these parameters to replace the learned per-channel affine transformation in a layer norm layer (Ba et al., 2016), which we insert immediately prior to the final linear layer in the DQN head.

We apply FiLM after the first layer in the DQN's MLP head. We parameterize our FiLM generator j as a small convolutional network, which takes the goal g (viewed as a $64 \times 7 \times 7$ spatial feature map) as input and applies two 128-channel convolutions followed by a flatten and linear layer to produce the FiLM parameters γ and β.

A.2.3 Model Architectures

In addition to the standard three-layer CNN encoder introduced by Mnih et al. (2015), we experiment with larger residual networks (He et al., 2016). We use the design proposed by Espeholt et al. (2018) as a starting point, while still adopting innovations used in more modern architectures such as EfficientNets (Tan and Le,

2019) and MobileNetv2 (Sandler et al., 2018). In particular, we use inverted residual blocks with an expansion ratio of 2, and batch normalization (Ioffe and Szegedy, 2015) after each convolutional layer. We use three groups of three residual blocks with 32, 64 and 64 channels each, downscaling by a factor of three in the first group and two in each successive group. This yields a final representation of shape $64 \times 7 \times 7$ when applied to 84×84-dimensional Atari frames, identical to that of the standard CNN encoder. In our scaling experiment with a larger network, we increase to five blocks per group, with 48, 96 and 96 channels in each group, as well as using a larger expansion ratio of 4, producing a representation of shape $96 \times 7 \times 7$. This enlargement increases the number of parameters by roughly a factor of 5. Finally, our DQN head has 512 hidden units, as opposed to 256 in SPR.

A.2.4 Image Augmentation

We use the same image augmentations as used in SPR (Schwarzer et al., 2021a), which itself used the augmentations used in DrQ (Kostrikov et al., 2021), in all experiments, including during both pretraining and fine-tuning. Specifically, we employ random crops (4 pixel padding and 84x84 crops) in combination with image intensity jittering.

A.2.5 Experiments with ATC

As ATC (Stooke et al., 2021) was not tested on the Atari100k setting, and as its hyperparameters (including network size and fine-tuning scheme) are very different from those used by SGI, we modify its code[1] to allow it to be fairly compared to SGI. We replace the convolutional encoder with that used by SGI, and use the same optimizer settings, image augmentation, pre-training data, and number of pre-training epochs as in SGI. However, we retain ATC's mini-batch structure (i.e., sampling 32 subsequences of eight consecutive time steps, for a total batch size of 512), as this structure defines the negative samples used by ATC's InfoNCE loss. During fine-tuning, we transfer the ATC projection head to the first layer of the DQN MLP head, as in SPR; we otherwise fine-tune identically to SGI, including using SPR.

1. https://github.com/astooke/rlpyt/tree/master/rlpyt/ul

A.3 Pseudocode

Algorithm 1: Pre-Training with SGI

Denote parameters of online encoder f_o, projection p_o and Q-learning head as θ_o;
Denote parameters of target encoder f_m, projection p_m and Q-learning target head as θ_m;
Denote parameters of transition model h, predictor q, inverse model I as ϕ;
Denote the maximum prediction depth as K, batch size as N;
Denote distance function in goal RL reward as d;
initialize offline dataset D;
while *Training* **do**
  sample a minibatch of sequences of $(s_t, a, s+t+1) \sim D$; `// sample unlabeled data`
  `/* sample goals */`
  for i *in range*$(0, N)$ **do**
    $s^i \leftarrow \text{augment}(s^i); s'^i \leftarrow \text{augment}(s'^i)$; `// augment input images`
    $j \sim$ Discrete Uniform$(1, 50)$; `// sample hindsight goal states`
    $g^i \leftarrow f_m(s^n_j)$; `// encode goal states`
    $\alpha \sim$ Uniform$(0, 0.5)$, $n \sim$ Normal$(0, 1)$; `// sample noise parameters`
    $g^i \leftarrow \alpha g^i + (1-\alpha)n$; `// apply noise`
    `/* Permute to make some goals very challenging to reach */`
    permute $\sim$ Bernoulli(0.2)
    if *permute* **then**
      $j \sim$ Discrete Uniform(N)
      $g^i \leftarrow g^j$; `// permute goal`
  `/* compute SGI loss */`
  for i *in range*$(0, N)$ **do**
    $\hat{z}^i_0 \leftarrow f_\theta(s^i_0)$; `// compute online representations`
    $l^i \leftarrow 0$;
    `/* compute SPR loss */`
    for k *in (1, . . . , K)* **do**
      $\hat{z}^i_k \leftarrow h(\hat{z}^i_{k-1}, a^i_{k-1})$; `// latent states via transition model`
      $\tilde{z}^i_k \leftarrow f_m(s^i_k)$; `// target representations`
      $\hat{y}^i_k \leftarrow q(p_o(\hat{z}^i_k))$, $\tilde{y}^i_k \leftarrow g_m(\tilde{z}^i_k)$; `// projections`
      $l^i \leftarrow l^i - \lambda^{\text{SPR}} \left(\frac{\hat{y}^i_k}{||\hat{y}^i_k||_2}\right)^\top \left(\frac{\tilde{y}^i_k}{||\tilde{y}^i_k||_2}\right)$; `// SGI loss at step` k
    `/* compute inverse modeling loss */`
    for k *in (1, . . . , K)* **do**
      $l^i \leftarrow \lambda^{\text{IM}} \cdot$ Cross-entropy loss$(a^i_{k-1}, I(\hat{y}_{k-1}, \tilde{y}_k))$
    `/* compute goal RL loss */`
    $r^i \leftarrow d(g^i, \tilde{z}_t) - d(g^i, \tilde{z}_{t+1})$; `// Calculate goal RL reward`
    $l^i \leftarrow l^i +$ RL loss(s^i, a^i, r^i, s'^i) ; `// Add goal RL loss for batch`
  $l \leftarrow \frac{1}{N}\sum_{i=0}^{N} l^i$; `// average loss over minibatch`
  $\theta_o, \phi \leftarrow \text{optimize}((\theta_o, \phi), l)$; `// update online parameters`
  $\theta_m \leftarrow \tau\theta_o + (1-\tau)\theta_m$; `// update target parameters`
return $(\theta_o,\ \phi)$; `// return parameters for fine-tuning`

A.4 Full Results on Atari100k

We report full scores for SGI agents across all 26 games in Table A.2. We do not reproduce the per-game scores for APT and VISR provided by Liu and Abbeel (2021), as we believe that the scores in the currently-available version of their paper may contain errors. [2]

Table A.2 – Mean return per episode for the 26 Atari100k games (Kaiser et al., 2019) after 100k steps. Agents are evaluated at the end of training, and scores for all methods are averaged over 10 random seeds. We reproduce scores for SPR from Schwarzer et al. (2021a), whereas ATC scores are from our implementation.

	Random	Human	SPR	ATC	SGI-R	SGI-E	SGI-W	SGI-C/S	SGI-C	SGI-C/L
Alien	227.8	7127.7	801.5	699.0	1034.5	857.6	1043.8	1070.5	1101.7	**1184.0**
Amidar	5.8	1719.5	176.3	95.4	154.8	166.8	**206.7**	185.9	168.2	171.2
Assault	222.4	742.0	571.0	509.8	446.6	583.1	759.5	632.4	905.1	**1326.5**
Asterix	210.0	8503.3	977.8	454.1	754.6	953.6	**1539.1**	651.8	835.6	567.2
Bank Heist	14.2	753.1	380.9	534.9	397.4	514.8	426.3	547.4	**608.4**	567.8
Battle Zone	2360.0	37187.5	**16651.0**	13683.8	4439.0	16417.0	7103.0	12107.0	13170.0	14462.0
Boxing	0.1	12.1	35.8	16.8	57.7	33.6	50.2	40.0	36.9	**73.9**
Breakout	1.7	30.5	17.1	16.9	23.4	17.8	35.4	23.8	42.8	**251.9**
Chopper Command	811.0	7387.8	974.8	870.8	784.7	1136.2	1040.1	1042.7	**1404.0**	1037.9
Crazy Climber	10780.5	35829.4	42923.6	74215.5	50561.2	76356.3	81057.4	75542.1	88561.2	**94602.2**
Demon Attack	152.1	1971.0	545.2	524.6	2198.7	357.5	1408.5	1135.5	968.1	**5634.8**
Freeway	0.0	29.6	24.4	5.7	2.1	15.1	26.5	12.5	**30.0**	28.6
Frostbite	65.2	4334.7	**1821.5**	222.6	349.3	981.4	247.7	861.1	741.3	927.8
Gopher	257.6	2412.5	715.2	946.2	1033.9	964.9	1846.0	1172.4	1660.4	**2035.8**
Hero	1027.0	30826.4	7019.2	6119.4	7875.2	6863.7	7503.9	7090.4	7474.0	**9975.9**
Jamesbond	29.0	302.8	365.4	272.6	263.9	383.8	**425.1**	413.2	366.4	394.8
Kangaroo	52.0	3035.0	**3276.4**	603.1	923.8	1588.9	598.6	1236.8	2172.8	1887.5
Krull	1598.0	2665.5	3688.9	4494.7	5672.6	4070.7	5583.2	**6161.3**	5734.0	5862.6
Kung Fu Master	258.5	22736.3	13192.7	11648.2	13349.2	11802.1	14199.7	16781.8	16137.8	**17340.7**
Ms Pacman	307.3	6951.6	1313.2	848.9	411.0	1278.3	1970.8	1519.5	1520.0	**2218.0**
Pong	-20.7	14.6	-5.9	-13.5	-3.9	4.2	4.7	**9.7**	7.6	7.7
Private Eye	24.9	69571.3	**124.0**	95.0	95.3	100.0	100.0	84.7	90.0	83.8
Qbert	163.9	13455.0	669.1	572.2	595.0	717.6	**855.6**	804.7	709.8	702.6
Road Runner	11.5	7845.0	14220.5	7989.3	5476.0	9195.2	18011.9	12083.5	**18370.2**	18306.8
Seaquest	68.4	42054.7	583.1	415.7	735.3	615.2	656.1	728.2	728.4	**1979.3**
Up N Down	533.4	11693.2	28138.5	84361.2	67968.1	63612.9	**84551.4**	42165.6	79228.8	46083.3
Median HNS	0.000	1.000	0.415	0.204	0.326	0.456	0.589	0.423	0.679	**0.755**
Mean HNS	0.000	1.000	0.704	0.780	0.888	0.838	1.144	0.914	1.149	**1.590**
#Games > Human	0	0	7	5	5	6	8	6	**9**	**9**
#Games > 0	0	26	**26**	**26**	25	**26**	**26**	**26**	**26**	**26**

2. In particular, we observed that VISR claimed to have a score below −21 on Pong, which is impossible with standard settings.

Table A.3 – Mean return per episode for the 26 Atari100k games (Kaiser et al., 2019) after 100k steps for versions of SGI with modified fine-tuning, as discussed in Section 4.5. Agents are evaluated at the end of training, and scores for all methods are averaged over 10 random seeds. We reproduce scores for SPR from Schwarzer et al. (2021a).

	Random	Human	SGI-None	Naive	Frozen	No SPR	Full SSL	SGI-C
Alien	227.8	7127.7	835.9	1049.3	**1242.8**	1060.7	1117.6	1101.7
Amidar	5.8	1719.5	107.6	133.6	147.7	154.2	**206.0**	168.2
Assault	222.4	742.0	657.7	752.1	869.2	756.3	**1145.2**	905.1
Asterix	210.0	8503.3	832.9	**1029.3**	433.1	575.5	603.1	835.6
Bank Heist	14.2	753.1	613.2	**726.5**	273.6	365.8	323.4	608.4
Battle Zone	2360.0	37187.5	13490.0	**15708.0**	11754.0	13692.0	11689.8	13170.0
Boxing	0.1	12.1	6.6	24.0	**61.5**	34.7	42.7	36.9
Breakout	1.7	30.5	12.1	29.3	34.0	43.0	**62.6**	42.8
Chopper Command	811.0	7387.8	1085.2	1081.2	916.5	925.5	965.8	**1404.0**
Crazy Climber	10780.5	35829.4	19707.6	55002.4	65220.0	69505.6	69052.0	**88561.2**
Demon Attack	152.1	1971.0	778.8	850.0	1329.4	981.7	**1783.8**	968.1
Freeway	0.0	29.6	17.2	28.1	24.4	13.2	10.9	**30.0**
Frostbite	65.2	4334.7	1475.8	662.1	1045.4	482.1	**1664.9**	741.3
Gopher	257.6	2412.5	438.2	626.1	**2214.1**	1561.7	1998.7	1660.4
Hero	1027.0	30826.4	6472.0	5538.3	6353.3	5249.6	**8715.4**	7474.0
Jamesbond	29.0	302.8	157.4	324.2	358.2	346.8	**407.6**	366.4
Kangaroo	52.0	3035.0	**3802.8**	3091.6	800.0	685.6	999.5	2172.8
Krull	1598.0	2665.5	3954.0	5202.7	**6073.7**	5722.8	5323.9	5734.0
Kung Fu Master	258.5	22736.3	7929.4	11952.2	**19374.6**	15039.8	18123.2	16137.8
Ms Pacman	307.3	6951.6	990.2	1276.4	1663.3	1753.3	**1779.3**	1520.0
Pong	-20.7	14.6	-4.4	-4.2	3.8	3.9	-0.1	**7.6**
Private Eye	24.9	69571.3	62.8	**385.9**	96.7	90.5	90.0	90.0
Qbert	163.9	13455.0	720.0	664.8	587.6	681.3	**3015.8**	709.8
Road Runner	11.5	7845.0	5428.4	14629.7	14311.9	17036.5	13998.2	**18370.2**
Seaquest	68.4	42054.7	577.8	509.0	1054.4	**1397.8**	989.4	728.4
Up N Down	533.4	11693.2	46042.6	48856.6	29938.4	**105466.9**	45023.5	79228.8
Median HNS	0.000	1.000	0.343	0.425	0.499	0.452	0.397	**0.679**
Mean HNS	0.000	1.000	0.565	0.849	0.971	1.114	1.011	**1.149**
#Games > Human	0	0	3	8	8	8	8	**9**
#Games > SPR	0	19	10	14	15	14	17	**20**

Table A.4 – Mean return per episode for the 26 Atari100k games (Kaiser et al., 2019) after 100k steps for various combinations of SGI's pretraining objectives, as discussed in Section 4.5. Agents are evaluated at the end of training, and scores for all methods are averaged over 10 random seeds.

	Random	Human	None	S	G	I	G+I	S+G	S+I	SGI
Alien	227.8	7127.7	835.9	278.7	964.3	1161.6	571.2	1172.3	**1203.0**	1101.7
Amidar	5.8	1719.5	107.6	37.8	54.8	198.1	58.0	**210.5**	175.4	168.2
Assault	222.4	742.0	657.7	517.9	512.3	868.1	567.2	813.5	820.3	**905.1**
Asterix	210.0	8503.3	832.9	292.6	416.1	475.6	431.8	506.3	648.5	**835.6**
Bank Heist	14.2	753.1	**613.2**	3.1	115.2	357.6	57.2	423.3	547.5	608.4
Battle Zone	2360.0	37187.5	13490.0	4665.0	3336.0	14807.0	3249.0	12528.0	**15491.0**	13170.0
Boxing	0.1	12.1	6.6	-21.8	12.5	40.1	-0.4	**42.9**	38.3	36.9
Breakout	1.7	30.5	12.1	0.9	2.1	24.1	3.2	41.0	41.6	**42.8**
Chopper Command	811.0	7387.8	1085.2	799.7	813.1	973.1	923.7	1097.2	978.3	**1404.0**
Crazy Climber	10780.5	35829.4	19707.6	243.3	17760.3	51203.9	581.0	66228.5	83995.4	**88561.2**
Demon Attack	152.1	1971.0	778.8	668.9	316.9	**1524.6**	756.4	1008.4	1286.6	968.1
Freeway	0.0	29.6	17.2	15.2	17.7	2.6	19.3	**30.5**	29.1	30.0
Frostbite	65.2	4334.7	**1475.8**	427.2	523.3	395.0	215.4	530.5	463.3	741.3
Gopher	257.6	2412.5	438.2	60.7	129.0	**1966.1**	99.0	1747.4	1778.7	1660.4
Hero	1027.0	30826.4	6472.0	2381.2	3590.2	7177.6	3998.7	**8251.2**	7366.2	7474.0
Jamesbond	29.0	302.8	157.4	41.8	236.0	373.1	183.6	365.6	**378.4**	366.4
Kangaroo	52.0	3035.0	**3802.8**	129.8	401.6	1041.4	222.6	830.8	760.2	2172.8
Krull	1598.0	2665.5	3954.0	720.1	1241.4	**5859.8**	1582.4	5778.8	5808.6	5734.0
Kung Fu Master	258.5	22736.3	7929.4	79.7	453.7	16914.7	686.2	**17825.1**	14681.9	16137.8
Ms Pacman	307.3	6951.6	990.2	418.7	528.5	1620.1	293.3	**1847.1**	1715.9	1520.0
Pong	-20.7	14.6	-4.4	-20.9	-20.4	-3.0	-21.0	0.9	1.7	**7.6**
Private Eye	24.9	69571.3	62.8	-20.7	89.4	**100.0**	12.7	98.2	**100.0**	90.0
Qbert	163.9	13455.0	**720.0**	201.0	277.4	706.5	215.2	650.5	601.9	709.8
Road Runner	11.5	7845.0	5428.4	780.3	5592.9	17698.4	2617.8	18229.4	17443.5	**18370.2**
Seaquest	68.4	42054.7	577.8	105.7	193.2	965.3	118.8	**1115.0**	792.1	728.4
Up N Down	533.4	11693.2	46042.6	892.2	4399.7	58142.0	1313.4	52772.9	39771.3	**79228.8**
Median HNS	0.000	1.000	0.343	0.009	0.060	0.411	0.029	0.512	0.629	**0.679**
Mean HNS	0.000	1.000	0.565	-0.054	0.181	0.943	0.098	1.004	0.978	**1.149**
#Games > Human	0	0	3	0	1	7	0	**9**	8	**9**
#Games > SPR	0	19	10	1	1	18	1	**20**	19	**20**

A.5 Transferring Representations between Games

One advantage of pretraining representations is the possibility of representations being useful across games. Intuitively, we expect better transfer between similar games so we chose five "cliques" of games with similar semantics and visual elements. The cliques are shown in Table A.5. We pretrain on a dataset of 750k frames from each game in a clique (i.e. 3M frames for a clique of 4) and finetune on a single game. To show whether pretraining on other games is beneficial, we compare to a baseline of pretraining on just the 750k frames from the single Atari 100k game we use for finetuning.

Our results in Table A.6 show that pretraining with the extra frames from the clique games is mostly unhelpful to finetune performance. Only Kangaroo shows a modest improvement, a few games show no difference in performance, and most games show a decrease in performance when pretraining with other games. We believe that Atari may not be as suitable to transferring representations as other domains, and previous work using Atari to learn transferable representations has also had negative results (Stooke et al., 2021). Though game semantics can be similar, we note that even small differences in rule sets and visual cues can make transfer difficult.

Table A.5 – Cliques of semantically similar games

space	Space Invaders, Assault, Demon Attack, Phoenix
pacman	MsPacman, Alien, Bank Heist, Wizard Of Wor
platformer	Montezuma Revenge, Hero, Kangaroo, Tutankham
top scroller	Crazy Climber, Up N Down, Skiing, Journey Escape
side scroller	Chopper Command, James Bond, Kung Fu Master, Private Eye

Table A.6 – Mean return per episode for clique games in Atari100k (Kaiser et al., 2019) after 100k steps. Agents are evaluated at the end of training, and scores for all methods are averaged over 10 random seeds. Games in the same clique are placed together.

Game	Single	Clique
Assault	738.5	554.1
Demon Attack	1171.8	695.0
Alien	1183.9	830.2
Bank Heist	448.8	303.0
Ms Pacman	1595.8	1352.1
Kangaroo	489.2	994.0
Crazy Climber	52036.0	21829.8
Up N Down	18974.7	13493.9
James Bond	397.6	325.4
Kung Fu Master	16402.6	16499.0
Chopper Command	933.6	854.6

Bibliography

Joshua Achiam. Spinning Up in Deep Reinforcement Learning. 2018. URL https://spinningup.openai.com/en/latest/. 11

Naman Agarwal, Brian Bullins, and Elad Hazan. Second-order stochastic optimization for machine learning in linear time, 2017. URL https://arxiv.org/abs/1602.03943. 27

Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. An optimistic perspective on offline reinforcement learning. In *ICML*, 2020. 49

Afshine Amidi and Shervine Amidi. Deep learning cheatsheets for stanford's cs 230, 2019. URL https://github.com/afshinea/stanford-cs-230-deep-learning. viii, 6

Dario Amodei, Danny Hernandez, Girish Sastry, Jack Clark, Greg Brockman, and Ilya Sutskever. AI and Compute, 2019. URL https://openai.com/blog/ai-and-compute/. 4

Ankesh Anand, Evan Racah, Sherjil Ozair, Yoshua Bengio, Marc-Alexandre Côté, and R Devon Hjelm. Unsupervised state representation learning in atari. In *NeurIPS*, 2019. 47, 48

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in neural information processing systems*, pages 5048–5058, 2017. 45

Anonymous. Deep reinforcement learning at the edge of the statistical precipice, 2021. Anonymous submission to NeurIPS 2021. Will be replaced with deanonymized version when available. 49, 60

Devansh Arpit, Stanislaw Jastrzebski, Nicolas Ballas, David Krueger, Emmanuel Bengio, Maxinder S. Kanwal, Tegan Maharaj, Asja Fischer, Aaron Courville, Yoshua Bengio, and Simon Lacoste-Julien. A closer look at memorization in deep networks, 2017. URL https://arxiv.org/abs/1706.05394. 8, 15

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. 7, 64

Philip Bachman, R Devon Hjelm, and William Buchwalter. Learning representations by maximizing mutual information across views. In *NeurIPS*, 2019. 47

Adrià Puigdomènech Badia, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskyi, Daniel Guo, and Charles Blundell. Agent57: Outperforming the atari human benchmark, 2020. URL https://arxiv.org/abs/2003.13350. 13

Yamini Bansal, Gal Kaplun, and Boaz Barak. For self-supervised learning, rationality implies generalization, provably, 2020. 28

Björn Barz and Joachim Denzler. Do we train on test data? purging cifar of near-duplicates. *Journal of Imaging*, 6(6):41, Jun 2020. ISSN 2313-433X. doi: 10.3390/jimaging6060041. URL http://dx.doi.org/10.3390/jimaging6060041. 2, 31, 38

Samyadeep Basu, Philip Pope, and Soheil Feizi. Influence functions in deep learning are fragile. *arXiv preprint arXiv:2006.14651*, 2020. 27

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47, 2013. 13

Richard Bellman. A markovian decision process. *Journal of Mathematics and Mechanics*, 6(5):679–684, 1957. ISSN 00959057, 19435274. URL http://www.jstor.org/stable/24900506. 11

Lucas Beyer, Olivier J. Hénaff, Alexander Kolesnikov, Xiaohua Zhai, and Aäron van den Oord. Are we done with imagenet?, 2020. 2

Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006. 3

John Bridle. Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann, 1990a. URL https://proceedings.neurips.cc/paper/1989/file/0336dcbab05b9d5ad24f4333c7658a0e-Paper.pdf. 3

John S. Bridle. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In Françoise Fogelman Soulié and Jeanny Hérault, editors, *Neurocomputing*, pages 227–236, Berlin, Heidelberg, 1990b. Springer Berlin Heidelberg. ISBN 978-3-642-76153-9. 3

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020a. URL https://arxiv.org/abs/2005.14165. 9

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020b. 41

Joy Buolamwini and Timnit Gebru. Gender shades: Intersectional accuracy disparities in commercial gender classification. In *Conference on fairness, accountability and transparency*, pages 77–91. PMLR, 2018. 27

Yuri Burda, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, and Alexei A. Efros. Large-scale study of curiosity-driven learning, 2018. 46, 47, 49

Tiffany Tianhui Cai, Jonathan Frankle, David J. Schwab, and Ari S. Morcos. Are all negatives created equal in contrastive instance discrimination?, 2020. URL https://arxiv.org/abs/2010.06682. 10

Víctor Campos, Pablo Sprechmann, Steven Stenberg Hansen, Andre Barreto, Charles Blundell, Alex Vitvitskyi, Steven Kapturowski, and Adria Puigdomenech

Badia. Coverage as a principle for discovering transferable behavior in reinforcement learning, 2021. URL https://openreview.net/forum?id=INhwJdJtxn6. 13, 41, 46, 48, 49, 50, 52

Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. Unsupervised learning of visual features by contrasting cluster assignments, 2020. 9, 28, 31

Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers, 2021. 31

Pablo Samuel Castro, Subhodeep Moitra, Carles Gelada, Saurabh Kumar, and Marc G. Bellemare. Dopamine: A Research Framework for Deep Reinforcement Learning. 2018. URL http://arxiv.org/abs/1812.06110. 51

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020a. 28, 31

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. *ICML*, 2020b. 9, 10, 41, 47, 55

Ting Chen, Simon Kornblith, Kevin Swersky, Mohammad Norouzi, and Geoffrey Hinton. Big self-supervised models are strong semi-supervised learners. In *NeurIPS*, 2020c. 47

Dami Choi, Christopher J. Shallue, Zachary Nado, Jaehoon Lee, Chris J. Maddison, and George E. Dahl. On empirical comparisons of optimizers for deep learning, 2020. URL https://arxiv.org/abs/1910.05446. 4

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, David Belanger, Lucy Colwell, and Adrian Weller. Rethinking attention with performers, 2021. URL https://arxiv.org/abs/2009.14794. 7

Brian Chu, Vashisht Madhavan, Oscar Beijbom, Judy Hoffman, and Trevor Darrell. *Best Practices for Fine-Tuning Visual Classifiers to New Domains*, pages 435–442. 11 2016. ISBN 978-3-319-49408-1. doi: 10.1007/978-3-319-49409-8_34. 57

Gilad Cohen, Guillermo Sapiro, and Raja Giryes. Dnn or k-nn: That is the generalize vs. memorize question, 2019. URL https://arxiv.org/abs/1805.06822. 15, 20

Will Dabney, André Barreto, Mark Rowland, Robert Dadashi, John Quan, Marc G Bellemare, and David Silver. The value-improvement path: Towards better representations for reinforcement learning. 2021. 45, 57

Thomas Degris and Joseph Modayil. Scaling-up knowledge for a cognizant robot. In *AAAI Spring Symposium on Designing Intelligent Robots: Reintegrating AI.*, 2012. 43

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *ACL*, 2019. 41

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2020. URL https://arxiv.org/abs/2010.11929. 6, 17

Rachit Dubey, Pulkit Agrawal, Deepak Pathak, Tom Griffiths, and Alexei Efros. Investigating human priors for playing video games. In *International Conference on Machine Learning*, 2018. 41

Gabriel Dulac-Arnold, Nir Levine, Daniel J. Mankowitz, Jerry Li, Cosmin Paduraru, Sven Gowal, and Todd Hester. An empirical investigation of the challenges of real-world reinforcement learning. 2020. 48

Gintare Karolina Dziugaite, Alexandre Drouin, Brady Neal, Nitarshan Rajkumar, Ethan Caballero, Linbo Wang, Ioannis Mitliagkas, and Daniel M. Roy. In search

of robust measures of generalization, 2021. URL https://arxiv.org/abs/2010.11924. 8

Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? 11(19):625–660, 2010. 9

Lasse Espeholt, Hubert Soyer, Rémi Munos, Karen Simonyan, Volodymyr Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. IMPALA: scalable distributed deep-rl with importance weighted actor-learner architectures. *CoRR*, abs/1802.01561, 2018. URL http://arxiv.org/abs/1802.01561. 51, 55, 64

Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. In *International Conference on Learning Representations*, 2018. 46

Vitaly Feldman. Does learning require memorization? a short tale about a long tail, 2019. URL https://arxiv.org/abs/1906.05271. 15, 26, 28, 37

Vitaly Feldman and Chiyuan Zhang. What neural networks memorize and why: Discovering the long tail via influence estimation, 2020. URL https://arxiv.org/abs/2008.03703. 15, 16, 17, 18, 19, 21, 27, 29, 30, 31, 32, 33, 37, 38

Jonathan Frankle, David J. Schwab, and Ari S. Morcos. Training batchnorm and only batchnorm: On the expressive power of random features in cnns, 2021. URL https://arxiv.org/abs/2003.00152. 7

Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, April 1980. ISSN 1432-0770. doi: 10.1007/BF00344251. URL https://doi.org/10.1007/BF00344251. 4

Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning, 2016. URL https://arxiv.org/abs/1506.02142. 3

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser,

and Connor Leahy. The pile: An 800gb dataset of diverse text for language modeling, 2020. URL https://arxiv.org/abs/2101.00027. 2

Robert Geirhos, Kantharaju Narayanappa, Benjamin Mitzkus, Matthias Bethge, Felix A. Wichmann, and Wieland Brendel. On the surprising similarities between supervised and self-supervised models, 2020. 28

Carles Gelada, Saurabh Kumar, Jacob Buckman, Ofir Nachum, and Marc G Bellemare. Deepmdp: Learning continuous latent space models for representation learning. *ICML*, 2019. 43, 47

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. URL http://proceedings.mlr.press/v9/glorot10a.html. 8

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL http://www.deeplearningbook.org. 4

Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, et al. Bootstrap your own latent: A new approach to self-supervised learning. In *NeurIPS*, 2020a. 9, 41, 47

Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H. Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, Bilal Piot, Koray Kavukcuoglu, Rémi Munos, and Michal Valko. Bootstrap your own latent: A new approach to self-supervised learning, 2020b. 10, 28, 31

Daniel Guo, Bernardo Avila Pires, Bilal Piot, Jean-bastien Grill, Florent Altché, Rémi Munos, and Mohammad Gheshlaghi Azar. Bootstrap latent-predictive representations for multitask reinforcement learning. In *ICML*, 2020. 47

Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, Bilal Piot, Bernardo A Pires, and Rémi Munos. Neural predictive belief representations. *ICML*, 2018. 43

Raia Hadsell, Dushyant Rao, Andrei A. Rusu, and Razvan Pascanu. Embracing change: Continual learning in deep neural networks. *Trends in Cognitive Sciences*, 2020. doi: https://doi.org/10.1016/j.tics.2020.09.004. URL http://www.sciencedirect.com/science/article/pii/S1364661320302199. 57

Steven Hansen, Will Dabney, Andre Barreto, David Warde-Farley, Tom Van de Wiele, and Volodymyr Mnih. Fast task inference with variational intrinsic successor features. In *ICLR*, 2020. 41, 42, 46, 48, 49, 52

Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. doi: 10.1038/s41586-020-2649-2. URL https://doi.org/10.1038/s41586-020-2649-2. 63

Ali Hassani, Steven Walton, Nikhil Shah, Abulikemu Abuduweili, Jiachen Li, and Humphrey Shi. Escaping the big data paradigm with compact transformers, 2021. URL https://arxiv.org/abs/2104.05704. 17

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015a. URL https://arxiv.org/abs/1512.03385. viii, 7, 8, 17

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, 2015b. URL https://arxiv.org/abs/1502.01852. 8

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 64

Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *CVPR*, 2020. 47

Olivier J Hénaff, Aravind Srinivas, Jeffrey De Fauw, Ali Razavi, Carl Doersch, SM Eslami, and Aaron van den Oord. Data-efficient image recognition with contrastive predictive coding. *arXiv preprint arXiv:1905.09272*, 2019. 9, 41, 47

Danny Hernandez and Tom B. Brown. Measuring the algorithmic efficiency of neural networks, 2020. URL https://arxiv.org/abs/2005.04305. 4

Danny Hernandez, Jared Kaplan, Tom Henighan, and Sam McCandlish. Scaling laws for transfer, 2021. 55

Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Gheshlaghi Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *AAAI*, 2018. 13, 44, 45

Matteo Hessel, Ivo Danihelka, Fabio Viola, Arthur Guez, Simon Schmitt, Laurent Sifre, Theophane Weber, David Silver, and Hado van Hasselt. Muesli: Combining improvements in policy optimization. *arXiv preprint arXiv:2104.06159*, 2021. 43

R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Phil Bachman, Adam Trischler, and Yoshua Bengio. Learning deep representations by mutual information estimation and maximization. *ICLR*, 2019. 47

Yu Huang and Yue Chen. Autonomous driving with deep learning: a survey of state-of-art technologies. *arXiv preprint arXiv:2006.06091*, 2020. 27

Like Hui and Mikhail Belkin. Evaluation of neural architectures trained with square loss vs cross-entropy in classification tasks, 2021. URL https://arxiv.org/abs/2006.07322. 3

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015. 7, 65

Yiding Jiang, Behnam Neyshabur, Hossein Mobahi, Dilip Krishnan, and Samy Bengio. Fantastic generalization measures and where to find them, 2019. URL https://arxiv.org/abs/1912.02178. 8

Ziheng Jiang, Chiyuan Zhang, Kunal Talwar, and Michael C. Mozer. Characterizing structural regularities of labeled data in overparameterized models, 2020. viii, ix, 27, 29, 30, 31, 33, 35, 36, 37

Łukasz Kaiser, Mohammad Babaeizadeh, Piotr Miłos, Błażej Osiński, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, et al. Model based reinforcement learning for atari. In *ICLR*, 2019. x, xi, 13, 46, 48, 50, 51, 67, 68, 69, 71

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020. 55

Kacper Piotr Kielak. Do recent advancements in model-based deep reinforcement learning really improve data efficiency?, 2020. URL https://openreview.net/forum?id=Bke9u1HFwB. 46

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. URL https://arxiv.org/abs/1412.6980. 4

Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions, 2017. 27, 59

Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly, and Neil Houlsby. Big transfer (bit): General visual representation learning, 2020. URL https://arxiv.org/abs/1912.11370. 7

Simon Kornblith, Honglak Lee, Ting Chen, and Mohammad Norouzi. What's in a loss function for image classification?, 2020. URL https://arxiv.org/abs/2010.16402. 3

Ilya Kostrikov, Denis Yarats, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. In *ICLR*, 2021. 13, 46, 48, 51, 57, 65

Alex Krizhevsky. Learning multiple layers of features from tiny images, 2009a. URL https://www.cs.toronto.edu/~kriz/cifar.html. 17

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009b. URL https://www.cs.toronto.edu/~kriz/cifar.html. 2, 31

Brenden M Lake, Tomer D Ullman, Joshua B Tenenbaum, and Samuel J Gershman. Building machines that learn and think like people. *Behavioral and brain sciences*, 40, 2017. 41

James Lee-Thorp, Joshua Ainslie, Ilya Eckstein, and Santiago Ontanon. Fnet: Mixing tokens with fourier transforms, 2021. URL https://arxiv.org/abs/2105.03824. 7

Timothée Lesort, Natalia Díaz-Rodríguez, Jean-Franois Goudou, and David Filliat. State representation learning for control: An overview. *Neural Networks*, 108, 2018. 44, 45, 47

Hao Li, Pratik Chaudhari, Hao Yang, Michael Lam, Avinash Ravichandran, Rahul Bhotika, and Stefano Soatto. Rethinking the hyperparameters for fine-tuning. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=B1g8VkHFPH. 57

Seppo Linnainmaa. The representation of the cumulative rounding error of an algorithm as a taylor expansion of the local rounding errors. Master's thesis, University of Helsinki, 1970. URL https://people.idsia.ch//~juergen/linnainmaa1970thesis.pdf. 4

Zachary C. Lipton. The mythos of model interpretability, 2016. 27

Hao Liu and Pieter Abbeel. Unsupervised active pre-training for reinforcement learning, 2021. URL https://openreview.net/forum?id=cvNYovr16SB. 41, 42, 46, 48, 49, 52, 54, 55, 67

Shengchao Liu, Dimitris Papailiopoulos, and Dimitris Achlioptas. Bad global minima exist and sgd can reach them, 2019. URL https://arxiv.org/abs/1906.02613. 15

Kevin Lu, Aditya Grover, Pieter Abbeel, and Igor Mordatch. Pretrained transformers as universal computation engines, 2021. URL https://arxiv.org/abs/2103.05247. 7

Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation, 2015. URL https://arxiv.org/abs/1508.04025. 6

Hartmut Maennel, Ibrahim Alabdulmohsin, Ilya Tolstikhin, Robert J. N. Baldock, Olivier Bousquet, Sylvain Gelly, and Daniel Keysers. What do neural networks learn when trained with random labels?, 2020. URL https://arxiv.org/abs/2006.10455. 15, 20

Bogdan Mazoure, Remi Tachet des Combes, Thang Doan, Philip Bachman, and R Devon Hjelm. Deep reinforcement and infomax learning. In *NeurIPS*, 2020. 47

Luke Melas-Kyriazi. Do you even need attention? a stack of feed-forward layers does surprisingly well on imagenet, 2021. URL https://arxiv.org/abs/2105.02723. 7

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540), 2015. xii, 12, 13, 48, 49, 51, 55, 64

Zachary Nado, Justin M. Gilmer, Christopher J. Shallue, Rohan Anil, and George E. Dahl. A large batch optimizer reality check: Traditional, generic optimizers suffice across batch sizes, 2021. URL https://arxiv.org/abs/2102.06356. 4

Yuri Nesterov. A method for solving a convex programming problem with convergence rate o (1/k2). In *Soviet Mathematics. Doklady*, volume 27, pages 367–372, 1983. xii, 4

Andrew Y Ng, Daishi Harada, and Stuart J Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, 1999. 64

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018. 47

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019. 63

Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *ICML*. PMLR, 2017. 46

Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron C Courville. Film: Visual reasoning with a general conditioning layer. In *AAAI*, 2018. 64

Geoff Pleiss, Tianyi Zhang, Ethan R. Elenberg, and Kilian Q. Weinberger. Identifying mislabeled data using the area under the margin ranking, 2020. URL https://arxiv.org/abs/2001.10528. 17

B. T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964. ISSN 0041-5553. doi: https://doi.org/10.1016/0041-5553(64)90137-5. URL https://www.sciencedirect.com/science/article/pii/0041555364901375. 4

Garima Pruthi, Frederick Liu, Mukund Sundararajan, and Satyen Kale. Estimating training data influence by tracing gradient descent, 2020. 27, 59

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision, 2021. 9, 31

Nitarshan Rajkumar, David Krueger, and Laurent Charlin. Does your model memorize where you think it does?, 2021. 27

Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do imagenet classifiers generalize to imagenet?, 2019. URL https://arxiv.org/abs/1902.10811. 31, 39

Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958. ISSN 1939-1471, 0033-295X. doi: 10.1037/h0042519. URL http://doi.apa.org/getdoi.cfm?doi=10.1037/h0042519. 4

Frank Rosenblatt. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. 1962. 4

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, October 1986. ISSN 1476-4687. doi: 10.1038/323533a0. URL https://www.nature.com/articles/323533a0. 4

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge, 2015. URL https://image-net.org/challenges/LSVRC/. 2, 31

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018. 65

Amartya Sanyal, Puneet K Dokania, Varun Kanade, and Philip H. S. Torr. How benign is benign overfitting?, 2020. URL https://arxiv.org/abs/2007.04028. 15

Robin M. Schmidt, Frank Schneider, and Philipp Hennig. Descending through a crowded valley – benchmarking deep learning optimizers, 2021. URL https://arxiv.org/abs/2007.01547. 4

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 2021. 55

Max Schwarzer. Data-efficient reinforcement learning with self-predictive representations. Master's thesis, Universite de Montreal, 8 2020. URL https://papyrus.bib.umontreal.ca/xmlui/handle/1866/25105. 11

Max Schwarzer, Ankesh Anand, Rishab Goel, R Devon Hjelm, Aaron Courville, and Philip Bachman. Data-efficient reinforcement learning with self-predictive representations. In *ICLR*, 2021a. x, xii, 13, 43, 44, 46, 47, 48, 51, 63, 65, 67, 68

Max Schwarzer, Nitarshan Rajkumar, Michael Noukhovitch, Ankesh Anand, Laurent Charlin, Devon Hjelm, Philip Bachman, and Aaron Courville. Pretraining

representations for data-efficient reinforcement learning, 2021b. URL https://arxiv.org/abs/2106.04799. xii, 41

Ramanan Sekar, Oleh Rybkin, Kostas Daniilidis, Pieter Abbeel, Danijar Hafner, and Deepak Pathak. Planning to explore via self-supervised world models. In *ICML*, 2020. 46

Alexander Selvikvåg Lundervold and Arvid Lundervold. An overview of deep learning in medical imaging focusing on mri. *arXiv e-prints*, pages arXiv–1811, 2018. 27

Archit Sharma, Shixiang Gu, Sergey Levine, Vikash Kumar, and Karol Hausman. Dynamics-aware unsupervised discovery of skills. In *International Conference on Learning Representations*, 2019. 46

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015. URL https://arxiv.org/abs/1409.1556. 17

Kihyuk Sohn, David Berthelot, Chun-Liang Li, Zizhao Zhang, Nicholas Carlini, Ekin D Cubuk, Alex Kurakin, Han Zhang, and Colin Raffel. Fixmatch: Simplifying semi-supervised learning with consistency and confidence. *arXiv preprint arXiv:2001.07685*, 2020. 47

Jiaming Song, Lunjia Hu, Michael Auli, Yann Dauphin, and Tengyu Ma. Robust and on-the-fly dataset denoising for image classification, 2020. URL https://arxiv.org/abs/2003.10647. 17

Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net, 2015. URL https://arxiv.org/abs/1412.6806. 6

Cory Stephenson, Suchismita Padhy, Abhinav Ganesh, Yue Hui, Hanlin Tang, and SueYeon Chung. On the geometry of generalization and memorization in deep neural networks, 2021. URL https://arxiv.org/abs/2105.14602. 15, 20

Adam Stooke and Pieter Abbeel. rlpyt: A research code base for deep reinforcement learning in pytorch. *arXiv preprint arXiv:1909.01500*, 2019. 63

Adam Stooke, Kimin Lee, Pieter Abbeel, and Michael Laskin. Decoupling representation learning from reinforcement learning, 2021. 42, 43, 47, 48, 49, 50, 52, 55, 65, 70

Ilya Sutskever. NVIDIA NTECH 2018 - Ilya Sutskever Keynote Talk, 2018. URL https://www.youtube.com/watch?v=w3ues-NayAs&t=1969s. 4

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html. 11

Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114, 2019. 64

Antti Tarvainen and Harri Valpola. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. In *NeurIPS*, 2017. 47

Pedro Tsividis, Thomas Pouncy, Jaqueline L. Xu, Joshua B. Tenenbaum, and Samuel J. Gershman. Human learning in atari. In *AAAI Spring Symposia*, 2017. 41

Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization, 2017. URL https://arxiv.org/abs/1607.08022. 7

Hado van Hasselt, Yotam Doron, Florian Strub, Matteo Hessel, Nicolas Sonnerat, and Joseph Modayil. Deep reinforcement learning and the deadly triad, 2018. URL https://arxiv.org/abs/1812.02648. 12

Hado P van Hasselt, Matteo Hessel, and John Aslanides. When to use parametric models in reinforcement learning? In *NeurIPS*, 2019. 13, 46, 51

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017. URL https://arxiv.org/abs/1706.03762. 6

Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity, 2020. URL https://arxiv.org/abs/2006.04768. 6

Tongzhou Wang and Phillip Isola. Understanding contrastive representation learning through alignment and uniformity on the hypersphere, 2020. 10, 56

Lilian Weng. Self-supervised representation learning. *lilianweng.github.io/lil-log*, 2019. URL https://lilianweng.github.io/lil-log/2019/11/10/self-supervised-learning.html. 9

Lilian Weng. Contrastive representation learning. *lilianweng.github.io/lil-log*, 2021. URL https://lilianweng.github.io/lil-log/2021/05/31/contrastive-representation-learning.html. 9

Paul J. Werbos. Applications of advances in nonlinear sensitivity analysis. In R. F. Drenick and F. Kozin, editors, *System Modeling and Optimization*, pages 762–770, Berlin, Heidelberg, 1982. Springer Berlin Heidelberg. ISBN 978-3-540-39459-4. 4

Yuxin Wu and Kaiming He. Group normalization, 2018. URL https://arxiv.org/abs/1803.08494. viii, 7

Lechao Xiao, Yasaman Bahri, Jascha Sohl-Dickstein, Samuel S. Schoenholz, and Jeffrey Pennington. Dynamical isometry and a mean field theory of cnns: How to train 10,000-layer vanilla convolutional neural networks, 2018. URL https://arxiv.org/abs/1806.05393. 8

Tete Xiao, Xiaolong Wang, Alexei A. Efros, and Trevor Darrell. What should not be contrastive in contrastive learning, 2021. URL https://arxiv.org/abs/2008.05659. 10

Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *ICML*, 2017. 57

Xiaohua Zhai, Alexander Kolesnikov, Neil Houlsby, and Lucas Beyer. Scaling vision transformers, 2021. URL https://arxiv.org/abs/2106.04560. 2

Albert Zhan, Philip Zhao, Lerrel Pinto, Pieter Abbeel, and Michael Laskin. A framework for efficient robotic manipulation, 2020. 48

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization, 2016. URL https://arxiv.org/abs/1611.03530. 8, 14

Chiyuan Zhang, Oriol Vinyals, Remi Munos, and Samy Bengio. A study on overfitting in deep reinforcement learning. *arXiv preprint arXiv:1804.06893*, 2018. 41

Roland S. Zimmermann, Yash Sharma, Steffen Schneider, Matthias Bethge, and Wieland Brendel. Contrastive learning inverts the data generating process, 2021. URL https://arxiv.org/abs/2102.08850. 10