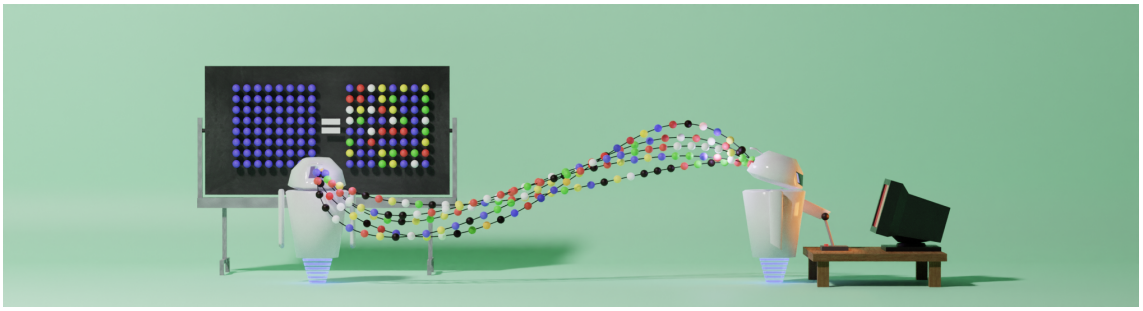
TÉCNICO LISBOA



Pretraining the Vision Transformer using self-supervised methods for vision-based deep reinforcement learning

Manuel Filipe Silva Goulão

Thesis to obtain the Master of Science Degree in

Computer Science and Engineering

Supervisor: Prof. Arlindo Manuel Limede de Oliveira

Examination Committee

Chairperson: Prof. Daniel Jorge Viegas Gonçalves
Supervisor: Prof. Arlindo Manuel Limede de Oliveira
Member of the Committee: Prof. Manuel Fernando Cabido Peres Lopes

November 2022

Acknowledgments

I would like to thank my supervisor, Prof. Arlindo Oliveira, for proposing the subject of this dissertation and guiding me through this journey, as well as for all the feedback and availability. I also wish to thank RNL (Rede e Laboratórios do DEI/IST) and INESC-ID for providing cluster services with several GPUs, which were essential to producing the results here presented. I am also grateful for the feedback provided by Bernardo Esteves and Pablo Samuel Castro, which positively impacted the quality of this work.

Abstract

The Vision Transformer architecture has shown to be competitive in the computer vision (CV) space where it has dethroned convolution-based networks in several benchmarks. Nevertheless, Convolutional Neural Networks (CNN) remain the preferential architecture for the representation module in Reinforcement Learning. In this work, we study pretraining a Vision Transformer using several state-of-the-art self-supervised methods and assess data-efficiency gains from this training framework. We propose a new self-supervised learning method called TOV-VICReg that extends VICReg to better capture temporal relations between observations by adding a temporal order verification task. Furthermore, we evaluate the resultant encoders with Atari games in a sample-efficiency regime, procgen games for measuring generalization and an imitation learning task for a fast and reliable comparison of the representations. Our data-efficiency results show that the vision transformer, when pretrained with TOV-VICReg, outperforms the other self-supervised methods and the non-pretrained vision transformer but still struggles to overcome a CNN. Our generalization results show some limitations in our method when used in more visually complex games which leads to degradation of the generalization performance. Nevertheless, we were able to outperform a CNN in two of the ten Atari games where we perform a 100k steps evaluation and show a consistent data-efficiency gain in comparison to the non-pretrained vision transformer. Ultimately, we believe that such approaches in Deep Reinforcement Learning (DRL) might be the key to achieving new levels of performance as seen in natural language processing and computer vision.

Keywords

Deep Reinforcement Learning; Transformers; Vision Transformer; Self-Supervised Learning; Pre-training; Atari.

Resumo

A arquitetura para tarefas de visão baseada em transformadores (vision transformer) tem mostrado ser competitiva na área de visão computacional (CV) onde tem destronado em enumeras *benchmarks* as redes baseadas em convoluções. No entanto, a rede neural convolucional (CNN) continua a arquitetura mais usada como módulo de representação em aprendizagem por reforço (RL). Neste trabalho, nós estudamos pré-treinar um vision transformer usando vários métodos estado-da-arte de aprendizagem auto-supervisionada com o intuito de avaliar ganhos na eficiência e capacidade de generalização por parte dos agentes que usam esses mesmos modelos. Nós propomos ainda um novo método auto-supervisionado ao qual chamamos TOV-VICReg que estende o método VICReg para melhorar a captura de informação temporal entre *frames* consecutivos. Os modelos pré-treinados são avaliados em termos de eficiência de amostras em vários jogos Atari e generalização em jogos do procgen. Os nossos resultados na eficiência de amostra mostram que o *vision transformer*, quando pré-treinado com o TOV-VICReg, consegue superar os restantes modelos pré-treinados mas ainda não consegue superar as CNN. Enquanto os nossos resultados na generalização mostram algumas limitações do nosso método quando usado em jogos visualmente mais complexos que levam a uma degradação na capacidade de generalizar. Mesmo assim, o nosso método foi capaz de superar as CNN em dois dos 10 jogos de Atari e obtemos um ganho consistente na eficiência de amostra em comparação com o *vision transformer* não pré-treinado. Em última análise, acreditamos que este tipo de abordagens em *deep reinforcement learning* poderão ser a chave para atingir novos níveis de performance como tem acontecido nas áreas de língua natural e visão computacional.

Palavras Chave

Aprendizagem por Reforço; Aprendizagem Profunda; Transformadores; Aprendizagem Auto-Supervisionada;

Pré-treino; Atari.

Contents

List of Figures

List of Tables

List of Algorithms

Listings

Acronyms

CNN Convolutional Neural Network

DRL Deep Reinforcement Learning

GAE Generalized Advantage Estimation

IQM Inter-Quartile-Mean

LSTM Long Short Term Memory

MDP Markov Decision Process

ML Machine Learning

MLP Multi-Layer Perceptron

RL Reinforcement Learning

RNN Recurrent Neural Networks

SOTA state-of-the-art

ViT Vision Transformer

1

Introduction

The Artificial Intelligence field has seen great successes in the last decade, in part due to the increase of computational capacity of the graphical processors (GPU), which led to the massive adoption of Deep Learning [1] methods that until then were too computationally expensive. Today we can use AI to generate photographs of people that don't exist, win against the world champion of Go, make diagnoses in medicine, and even automatically generate news articles. One of the areas responsible for some of those amazing applications and where, naturally, a lot of research is being made is called Reinforcement Learning.

In Reinforcement Learning (RL) the problem is not about learning labels or hidden patterns, but instead, about having an agent solve sequential decision problems by performing the best possible actions. RL, and more recently Deep Reinforcement Learning (DRL), demonstrate a large potential to solve problems that are currently impossible or very difficult for current Machine Learning algorithms. Successes in robotics [2, 3], board games [4], and control problems [5, 6], validate this idea. However, RL is still far from the promised impactful applications. For example, the work by Dulac-Arnold and colleagues [7] presents nine challenges that need to be addressed to make the application of RL in the real world more feasible. One of the challenges is "Learning on the real system from limited samples". Current state-of-the-art (SOTA) algorithms require in some environments millions of examples to start performing reasonably well while we Humans only need a few examples. In environments where the agent doesn't have full access to the current state (partially observable environments), this problem becomes even more prominent, since the agent not only needs to learn the state-to-action mapping but also a state representation function that tries to be informative about a state given an observation. In contrast, humans, when learning a new task, already have a well-developed visual system and a good model of the world which are components that allows us to easily learn new tasks. Previous works have tried to tackle the sample inefficiency problem by using auxiliary learning tasks [8–10], that try to help the network's encoder to learn good representations of the observations given by the environments. These tasks can be supervised or unsupervised and can happen during a pretraining phase or a reinforcement learning (RL) phase in a joint-learning or decoupled-learning scheme.

In recent years, self-supervised learning has shown to be very useful in computer vision. The increasing interest in this area has resulted in the appearance of new and improved methods that train a network to learn important features from the data using only the data itself as supervision. To evaluate and compare such methods a linear layer is trained on a certain task, like ImageNet, using as input the representations computed by a pretrained encoder. The results show high scores in different benchmarks, which shows how well the current state-of-the-art methods can encode useful information from the given images without being task-specific. Additionally, it has been shown that pretraining a network using self-supervised learning (or unsupervised learning) adds robustness to the network and gives better generalization capabilities [11].

Also recently, a new architecture for vision-based tasks called the Vision Transformer (ViT) [12] has shown impressive results in several benchmarks without using any convolutions. This architecture presents much weaker inductive biases when compared to a Convolutional Neural Network (CNN), which can result in lower data efficiency. However, the Vision Transformer, unlike the CNNs, can capture relations between parts of an image (patches) that are far apart from each other, thus deriving global information that can help the model perform better in certain tasks. Furthermore, when the model is pretrained, using supervised or self-supervised learning, it manages to surpass the best convolution-based models in terms of task performance. Despite these successes in computer vision these results are yet to be seen in reinforcement learning.

Motivated by the potential of the Vision Transformer, in particular when paired with a pretraining phase, and the increasing interest in self-supervised tasks for DRL, we study pretraining ViT using SOTA self-supervised learning methods and use it as the representation module in the DRL algorithm. Consequently, we propose extending VICReg (Variance Invariance Covariance Regularization) [13] with a temporal order verification task [14] to help the model better capture the temporal relations between consecutive observations. We named this approach Temporal Order Verification-VICReg or in short TOV-VICReg. While we could have adapted any of the other methods, we opted for VICReg due to its computational performance, simplicity, and good results in early experiments and metrics such as the ones presented in Section 6.1.

We study this training framework in three sections: Chapter 5, where we assess data-efficiency gains in Atari games when using a Vision Transformer pretrained with self-supervised learning methods; Chapter 6, where we evaluate the representations computed by the pretrained models through metrics, visualizations, and an evaluation task based on imitation learning and linear probing; and Chapter 7, where we assess if agents using a vision transformer pretrained with our method can better generalize to unseen levels using the procgen [15] games.

The results here presented were submitted and accepted at the 36th Conference on Neural Information Processing Systems (NeurIPS 2022) Deep Reinforcement Learning Workshop.

2

Background

Contents

2.1 Reinforcement Learning

Reinforcement Learning is an area of Machine Learning that tries to make an agent learn to solve a certain task by interacting with the environment. It differs from Supervised Learning since it doesn't use labeled datasets to learn the tasks, and also from Unsupervised Learning because the agent is not trying to find hidden structures but trying to maximize the rewards received.

The problem of an *agent* learning to solve the task in a certain *environment* can be defined as a Markov Decision Process (MDP). A MDP $\mathcal{M}$ is defined by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T} \rangle$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ the set of actions, $\mathcal{R}$ the reward function, and $\mathcal{T}$ the transition function. At each timestep the agent is in a state $s \in \mathcal{S}$ and takes an action $a \in \mathcal{A}$. Upon performing the action the agent receives from the environment a reward $r \in \mathcal{R}$ and a new state $s' \in \mathcal{S}$ which is determined by the transition function $\mathcal{T}(s', s, a)$. Figure 1 shows this exact interaction loop. The MDP assumes that the Markov property holds in the environment, i.e. the state transitions are independent and the agent only needs to know the current state to perform an action $P(a_t|x_0, x_1...x_t) = P(a_t|x_t)$.

For the agent to decide what action to take it uses a policy function π, which gives a distribution over actions given a state, $a_t = \pi(s_t)$. This policy is evaluated using the function $V^{\pi}(s)$, which estimates the expected total discounted reward of an agent in a state s and which follows a policy π, or the function $Q(s, a)$ given state-action pair.

A policy π is better than π' if and only if $v_{\pi}(s) \geq v_{\pi'}(s),\ \forall s \in \mathcal{S}$. The policy that has a value greater than or equal to all the other possible policies is called the optimal policy. Since the goal of Reinforcement Learning is to develop agents that perform in an optimal or nearly optimal way when given a certain MDP we can define the RL problem as the maximization of the value function $v_{\star}(s) = max_{\pi} v_{\pi}(s)$. The case here considered is the finite-horizon model which is very simple but sufficient to demonstrate the base idea [16, 17].

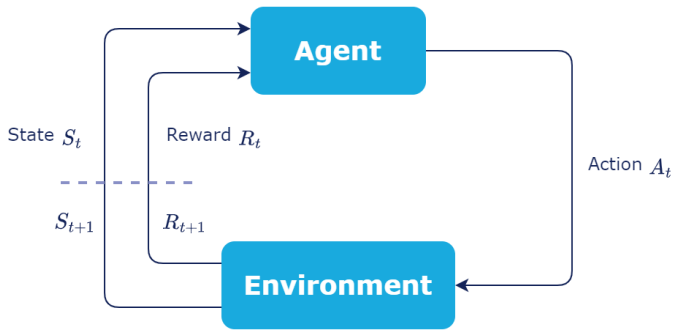


Figure 2.1: Reinforcement Learning interaction loop

2.1.1 Online Reinforcement Learning -

When we talk about Reinforcement Learning we are referring to what is called Online Reinforcement Learning. In Figure 2.2 we can observe an example of a learning loop where the agent interacts with the environment for a certain amount of time which is called a rollout and then uses the data collected in that rollout to update the current policy π_k to a new policy π_{k+1}. Note that this is just a simple example and different algorithms might have different learning loops.

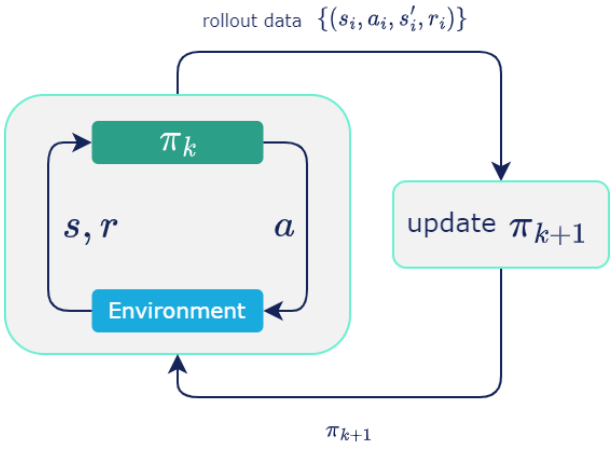


Figure 2.2: Online Reinforcement Learning learning loop

2.1.2 Offline Reinforcement Learning -

In many cases learning from scratch while interacting with the environment is just too expensive, making Online RL unfeasible in many applications. Offline Reinforcement Learning, also called Batch RL, on the other hand, doesn't rely on the interaction loop with the environment, instead, the policy is learned from a dataset with many transitions related to the task the agent is trying to solve. The goal in Offline RL is that the policy learned is equal to or better than the policy used to collect the data. Note that in some cases, after learning the policy, some fine-tuning might be needed, i.e. the agent improves the policy by interacting with the environment. Figure 2.3 shows an example of the learning scheme [18].

2.2 Deep Reinforcement Learning

The successes of Deep Learning methods in Supervised Learning quickly transferred to the Reinforcement Learning realm. In the last decade, we have witnessed the rise of Deep Learning based RL algorithms capable of solving problems that until then were considered impossible [4]. These successes can be explained by the capabilities of deep neural networks to find features from high-dimensional data

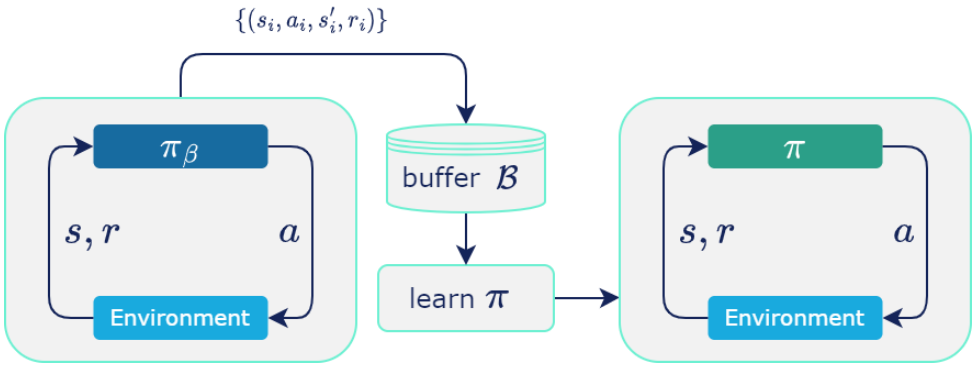


Figure 2.3: Offline Reinforcement Learning learning loop

with no previous knowledge.

Although these successes are impressive and exciting, other problems have arisen. One main problem is the sample efficiency of deep neural networks. Neural networks start learning from *tabula rasa* and need, in many problems, millions of examples to start performing well. As an example, the algorithm Muzero, a successor of AlphaGo, trains for 12 hours in board games while using 16 TPUs for training and 1000 TPUs for selfplay [19].

However, neural networks are not the only limitation in terms of sample efficiency. The RL algorithm itself is where a lot of work can be done to improve how fast the agent learns to solve a task. EfficientZero [20], for example, proposes three main changes to Muzero that significantly improve sample efficiency without losing general performance and none of those changes is related to the models used. A lot of work is being done to improve RL agents in different types of environments and tasks: **Imitation Learning**: the agent learns with an expert demonstrating some kind of behaviour that we want the agent to replicate; **Intrinsic Motivation**: extends the classic concept of reward with the introduction of intrinsic rewards, in an attempt to force the agent to explore, be curious, and learn by itself, especially in environments with few or no extrinsic rewards (the normal reward) [21]; **Model-Based**: usually the agents try to learn 3 functions: representation, dynamics, and behaviour which the agent can use to plan a good sequence of actions to make better decisions; **Hierarchical RL**: has been explored for several decades using frameworks like the Options and Feudal [22, 23]. More recently these frameworks have seen the introduction of neural networks with the FeUdal Networks [24] and the Option-Critic papers [25]; **Meta-RL** is concerned with the development of agents that can rapidly adapt to new tasks that they never encountered during training. A Meta-RL policy is very similar to the normal RL, but it considers not only the current state but also the previous state and previous action. The idea is for the agent to get an idea of the history which will help it adapt to unseen circumstances.

Although sometimes even without changing the algorithm or the models, we can obtain significant performance improvements like in the case of agents that use a Replay Buffer, where just changing the

way the saved transitions are sampled can make a huge difference. For example, the Prioritized Replay Buffer [26] changes the probability of a transition being sampled according to how relevant, in terms of learning, each trajectory is. This simple change doesn't affect the core algorithm of the agent but can drastically improve the agent performance.

2.2.1 DQN and Rainbow

DQN [27] is one of the most famous and influential Deep RL algorithms. The algorithm is a very straightforward value-based method. We have a network with parameters ϕ that given a state s outputs a prediction of the distribution of Q values over actions, $Q_\phi(s, a)$. To update the network the following equation is used as target: $y = r + \gamma \ max_{a'} Q_\phi(s', a')$ and compute the mean squared error: $(y - Q_\phi(s, a))^2$. Algorithm 2.1 shows the pseudocode for DQN. Figure 2.4 shows a simplified diagram of the DQN structure and functioning. Several works followed the DQN algorithm and introduced changes to improve its stability, data-efficiency, and general performance. Rainbow [28] combines six of those improvements. **Double Q-Learning** [29] introduces a second estimator of the Q function to avoid the problem of overestimating the action values; **Prioritized Experience Replay** [26] introduces a new experience replay which samples important transitions more frequently; **Dueling Networks** [30] that separates the last linear layers into two separate estimators: state value function and advantage function; **Multi-step Learning** [16] that accumulates the rewards across n-steps and uses the $n + 1$ greedy action to bootstrap the Q value when calculating the Q target; **Distributional RL** [31] changes the model to predict a discrete distribution over values instead of a single value; and **Noisy Nets** [32] that replaces the normal linear layers of the model with noisy linear layers, which are composed by a normal linear layer stream and a noisy stream, allowing for the model to explore more in the first iterations and removing the need for an ϵ-greedy policy. All these changes combined result in an algorithm that is more stable and sample efficient. Even though it was proposed in 2017, Rainbow is still one of the state-of-the-art algorithms in the category of Model-free Reinforcement Learning.

Algorithm 2.1: DQN algorithm

for $episode \leftarrow 1$ **to** M **do**
  for $t \leftarrow 1$ **to** T **do**
    With probability ϵ: $a_t = random()$, otherwise: $a_t = argmax_{a'} \ Q(s, a')$;
    Execute a_t and observe s'_t and r_t;
    Store transition $\{s_t, a_t, r_t, s'_t\}$ in the replay buffer $\mathcal{D}$;
    Sample a mini-batch of transitions $\{s_j, a_j, r_j, s'_j\}$ from $\mathcal{D}$;
    $y_j = r_j + \gamma \ max_{a'_j} Q_\phi(s'_j, a'_j)$;
    $\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi(s_j, a_j)}{d\phi}(Q_\phi(s_j, a_j) - y_j)$;

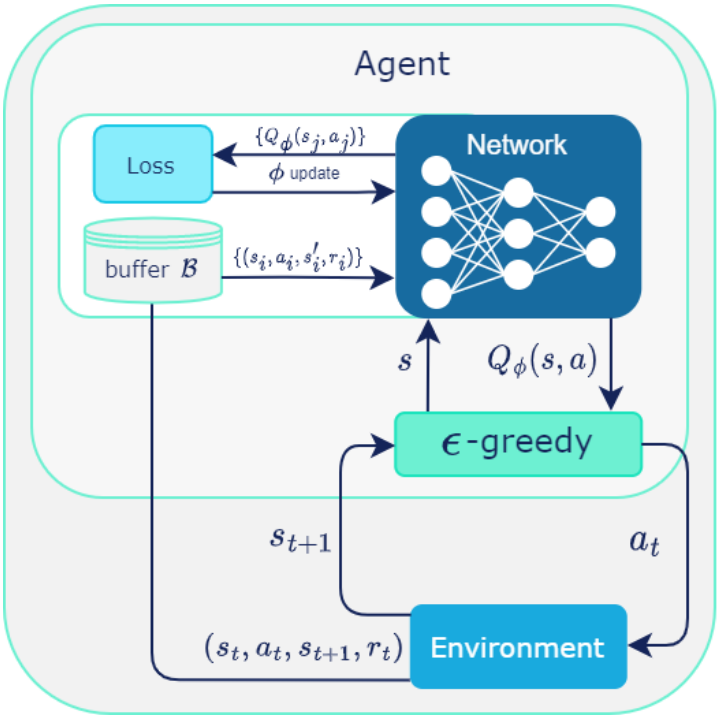


Figure 2.4: DQN

2.2.2 PPO

PPO [33] is an on-policy algorithm from the family of policy gradient methods which learns to perform a task by maximizing the expected return using stochastic gradient ascent [17].

In the case of a simple policy gradient consider that we have a policy π_θ, where θ denotes the parameters of the network that computes the policy function, and we want to maximize the expected return $J(\pi_\theta) = \mathbb{E}_{\tau\sim\rho_\theta}[R(\tau)]$, where ρ_θ is the probability distribution over trajectories given some parameter θ. We can then define the goal as being the search for a θ that maximizes J, $\theta^\star = \text{argmax}_\theta J(\pi_\theta)$, using stochastic gradient ascent the optimization step can be defined by Equation 2.1 (a complete derivation of J can be found in Appendix A.1).

$$\begin{aligned}\theta_{t+1} &= \theta_t + \alpha \nabla_\theta J(\pi_\theta)|_{\theta_t} \\ &= \theta_t + \alpha \mathop{\mathbb{E}}_{\tau\sim\rho_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau)\right]\end{aligned} \tag{2.1}$$

This expectation can be estimated using the arithmetic mean, Equation 2.2

$$\nabla_\theta J(\pi_\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(s_{i,t}|a_{i,t})\right)\left(\sum_{t=1}^{T} r(s_{i,t}, a_{i,t})\right) \tag{2.2}$$

A simple policy gradient algorithm that uses this estimation to optimize the policy is the REINFORCE algorithm 2.2.

Algorithm 2.2: REINFORCE algorithm

```
Initialize network parameters;
Initialize memory;
for i ← 0 to S do
    Collect trajectories using π_θ;
    for j ← 0 to B do
        Sample action a_t from policy π_θ;
        Execute a_t and observe s'_t and r'_t;
        Save transition into memory;
    θ ← θ + α (1/N) Σ_{i=1}^{N} ( Σ_{t=1}^{T} ∇_θ log π_θ(s_{i,t}|a_{i,t}) ) ( Σ_{t=1}^{T} r(s_{i,t}, a_{i,t}) );
```

While PPO can be considered a policy gradient method it actually falls in a subset of algorithms called policy iteration. These methods use an objective function, usually called surrogated advantage, that measures a certain policy performance relative to an old policy by maximizing the trajectory of a new policy using the advantage estimator trained in the previous policy, Equation 2.3. The resulting loss function can be found in Equation 2.4 and a complete derivation of both equations at Appendix A.2. PPO uses this objective function and applies a clipping that discourages the new policy from getting too far from the old policy, Equation 2.5, where D is the set of trajectories. To estimate the advantage function an additional network is used, which computes the state values by optimized the loss function in Equation 2.6, where $\hat{R}_t$ is the reward-to-go at time t.

$$\text{argmax}_\theta J(\pi_{\theta'}) - J(\pi_\theta) = \text{argmax}_\theta \mathop{\mathbb{E}}_{\tau \sim \rho_{\theta'}} \left[\sum_t \gamma^t A^{\pi_\theta}(s_t, a_t) \right] \tag{2.3}$$

$$\mathcal{L}(\theta', \theta) = \mathop{\mathbb{E}}_{s,a \sim \pi_\theta} \frac{\pi_{\theta'}(a|s)}{\pi_\theta(a|s)} A^{\pi_\theta}(s, a) \tag{2.4}$$

$$\mathcal{L}(s, a, \theta', \theta) = \frac{1}{|D|\,T} \sum_{\tau \sim \mathcal{D}} \sum_{i=0}^{T} \text{min} \left(\frac{\pi_{\theta'}(a|s)}{\pi_\theta(a|s)} A^{\pi_\theta}(s, a), \text{clip}(\frac{\pi_{\theta'}(a|s)}{\pi_\theta(a|s)}, 1-\epsilon, 1+\epsilon) A^{\pi_\theta}(s, a) \right) \tag{2.5}$$

$$\mathcal{L}(s, \phi) = \frac{1}{|D|\,T} \sum_{\tau \sim \mathcal{D}} \sum_{i=0}^{T} (V_\phi(s_t) - \hat{R}_t)^2 \tag{2.6}$$

Current SOTA PPO implementations also take advantage of several other implementation details that help the model achieve a better performance. This includes Generalized Advantage Estimation (GAE) [34], orthogonal initialization of weights and constant initialization of biases [35], value loss clipping, entropy bonus term, and advantages normalization.

2.3 Transformers

2.3.1 Attention

The American Psychological Association defines attention as ”a state in which cognitive resources are focused on certain aspects of the environment rather than on others and the central nervous system is in a state of readiness to respond to stimuli.” [36]. When the attention mechanism was first introduced in the context of Deep Learning [37] it was used in an RNN encoder-decoder framework for natural language translation. The authors used the attention mechanism in the decoder so the encoder didn't need to encode all the information in the source. Further work [38] developed the concept of attention with the introduction of the concepts of local and global attention and the study of different score functions.

2.3.2 Self-Attention

Intra-attention or self-attention [39] is a mechanism that relates different positions of a sequence to create representations between the different tokens in that sequence. It has been, for example, used in machine translation where the system learns correlations between the current word and the others which allows it to pay more attention to the more important words.

2.3.3 Transformer

The Transformer [40] is an architecture that solely uses attention mechanisms removing entirely the Recurrences and Convolutions largely used until that point. Before the Transformer, we would probably use a seq2seq model (Encoder-Decoder) using Recurrent Neural Networks (RNN) or a Long Short Term Memory (LSTM) network. These models process the input sequentially, which is a problem when we have long sequences since these models start losing information from old inputs, and for that reason, the encoded information stops having information from words it might have seen some timesteps ago [41]. The Transformer on the other hand doesn't compute the input sequentially, it takes the complete input and takes advantage of the self-attention mechanism to focus on the most important parts. Another advantage of the Transformer is the explainability it introduces compared to the RNNs and LSTMs, since we can easily analyze the learned relations between tokens and use them as a good explanation for the output obtained.

The canonical Transformer model architecture, in Figure 2.5, has two main components: an encoder and a decoder. The encoder receives an input, e.g. a complete sentence in English, and creates a latent representation of this input, while the decoder receives that latent representation and the ground-truth output shifted right, i.e. the tokens are shifted one position to the right and the first token becomes the beginning of sentence token (<BOS>) and outputs a value that will feed a linear layer. The output of

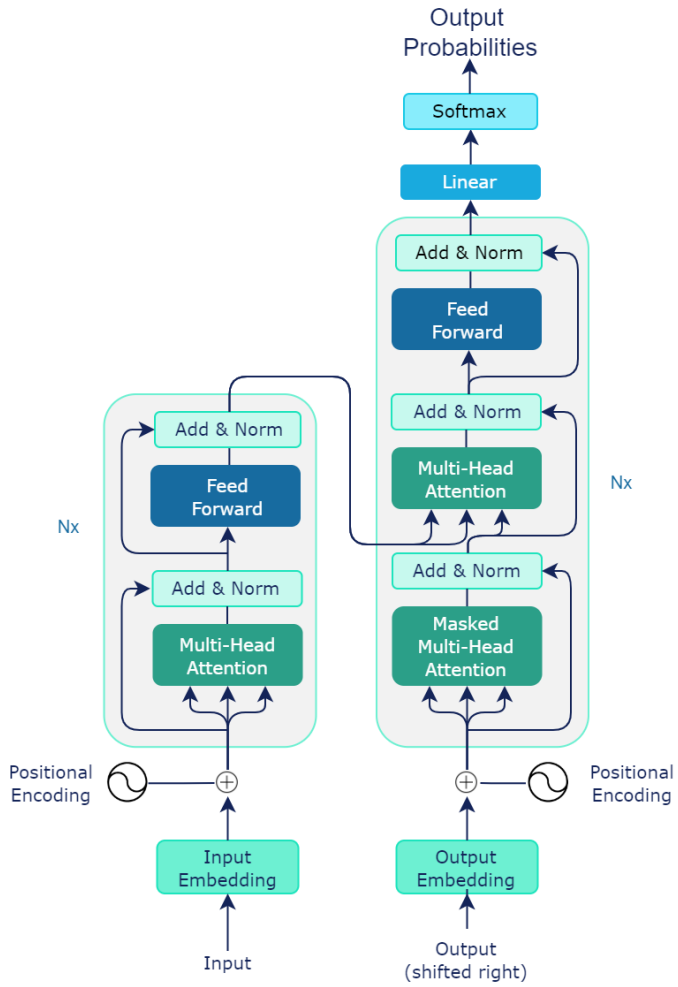


Figure 2.5: The Transformer model architecture

the linear layer after normalisation using a softmax is used as the probability of a set of words being the next token in the sentence.

Inside both the encoder and decoder, we can find the so-called Multi-Head Attention layer. This layer is the composition of several heads, where each head is computing attention in parallel (Figure 2.6 and Equation 2.7). The capacity of the Multi-Head Attention layers to "jointly attend to information from different representation subspaces at different positions" [40] in parallel is the reason why Transformers models are so fast and scalable.

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = [\text{head}_1; \ldots; \text{head}_h]\mathbf{W}^O \tag{2.7}$$

The attention computed inside each head is the scaled dot-product with a scaling factor, Figure 6.4 and Equation 2.8, where Q, K, and V are called the Query, Key, and Value and are defined as follows: $q_i = x_i W^Q, k_i = x_i W^K, v_i = x_i W^V$ where q_i, qk_i and v_i are the i-th row of the matrices Q, K, and V,

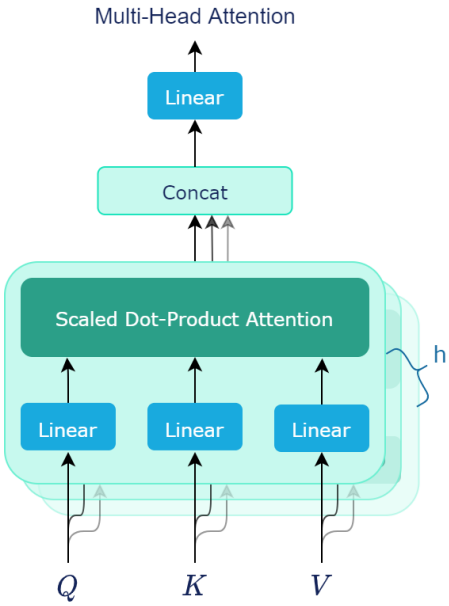


Figure 2.6: Multi-Head Attention layer with h heads

respectively.

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{n}})\mathbf{V} \tag{2.8}$$

The Q, K, and V notation is inspired by the information retrieval systems. In these systems, we have a dataset where each entry has two columns: Key, and Value. When we perform a query, the value of the query is compared against multiple keys. If we imagine that the queries and keys are vectors with equal dimensions, then one way of comparing those vectors would be (just like self-attention) with a dot-product operation. And consequently, the result of that operation would allow us to obtain a weighted map of the most relevant values.

2.3.4 Vision Transformer

The Vision Transformer, or ViT, [12] (Figure 2.8) is a model for image classification tasks that doesn't rely on CNNs. The architecture was designed to be as close as possible to the original Transformer making only changes where necessary. A major change is how the input is handled because processing pixels as tokens would be unfeasible, the vision transformer instead splits the image into patches. The patches are projected into an embedding vector (usually using a learnable projector, for example, a convolution followed by a flatten operation) and placed in a linear sequence. Then, a representation token embedding is concatenated to the sequence, a positional encoding is added to each embedding and then the resulting sequence is fed into the Transformer. For image classification tasks, this vector is

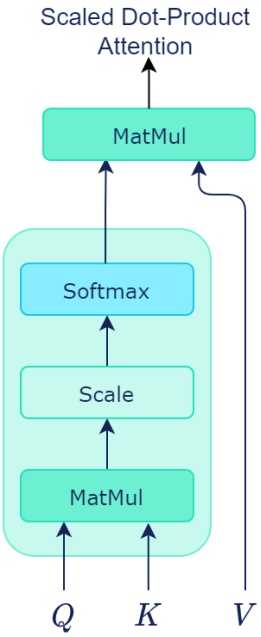


Figure 2.7: Scaled Dot-product Attention

then fed into an Multi-Layer Perceptron (MLP) that will compute the class probabilities. The representation token is used in order for the complete model (ViT and MLP) to be able to classify an image, using the representation of the image, which corresponds to the representation token embedding after being computed by ViT. When compared to CNNs, ViT presents weaker image-specific inductive biases which can impact the sample-efficiency of the model during learning [42], however, it has been shown that with enough data the image-specific inductive biases become less important [12]. Moreover, ViT can capture relations between patches that are far apart from each other, thus deriving global information that can help the model perform better in certain tasks

2.3.5 Other Variations

The results shown by the Transformer created a big interest from the research community in proposing new variations that can work in different domains including RL. The Transformer-XL [43] introduces a recurrence mechanism for state reuse to maintain information after each computation, and so be able to make recurrent computations. The Feedback Transformer [44] is an autoregressive model that targets the limitations of the canonical Transformer in sequential token prediction tasks. This architecture processes a sequence sequentially instead of the parallel way found in the canonical Transformer. While this won't allow having the computational gains that parallel processing has, the authors argue that the Feedback Transformer will instead be able to exploit "the sequential nature of the input".

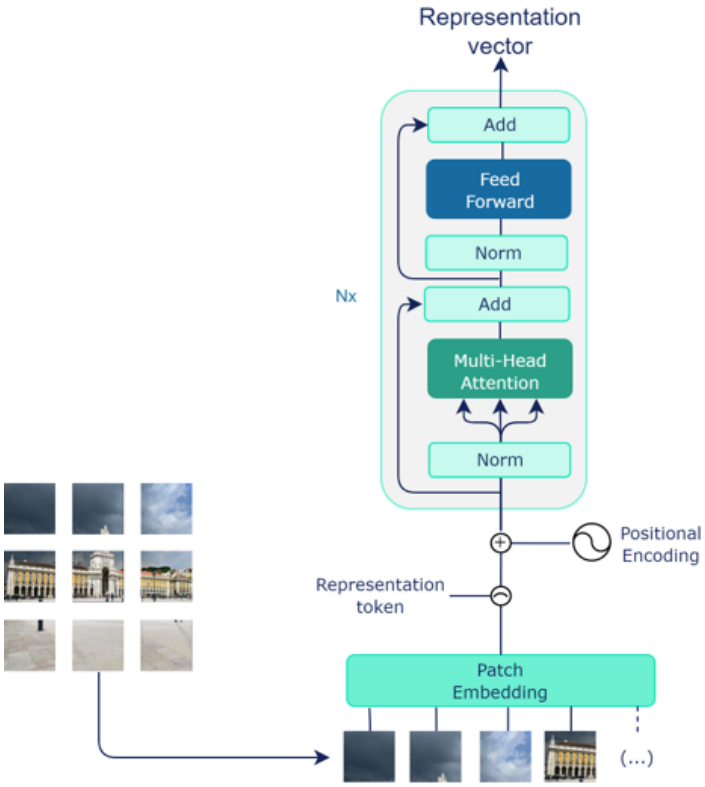


Figure 2.8: ViT architecture

2.4 Self-Supervised Learning

The term unsupervised learning was used to encapsulate some of the methods that we currently call self-supervised learning. The transition to this new term is mostly due to the ambiguity of the word since it suggested that there was no supervision which is not true. Self-supervised learning methods, as the name better suggests, are trained using the data as supervision [45]. This area of Machine Learning creates an opportunity to solve problems where the annotation of the data is expensive or where we just want to use a big dataset of unlabelled data. Moreover, it allows us to learn good representations, which can be used to bootstrap a model that will be fine-tuned in a different task. Furthermore, the previous results indicate that self-supervised pre-training can lead to better generalization in neural networks [11], so the successes of models like BERT [46] might, in time, be seen in other areas of Machine Learning (ML).

The field has seen in recent years an increase in interest and subsequently a fast growth in published work and methods. Especially with the rise of more expressive models like the Transformer, the use of self-supervised learning in pre-training became common practice. Currently, we can consider the following big categories of self-supervised tasks (also called pretext tasks): **Generative:** the model learns a good representation of the training data to better generate new samples of data [47]; **Self-prediction:**

instead of predicting the complete sample like the Generative methods self-prediction methods only predict parts of the sample which have been masked [46]; **Innate Relationships:** these tasks consist in learning relations between parts of the same sample, for example, in the case of an image we can split it into several patches, swap the patches' positions and make the model predict the original position of each patch [48, 49]; **Contrastive:** consists in the process of learning to discriminate between two inputs that might be similar (positive) or different (negative) [50, 51]; **Non-Contrastive:** these methods are somewhat similar to contrastive methods, especially if we consider that most methods in these two categories usually use a siamese network. However, non-contrastive methods don't have the notion of negative samples. Instead, they approximate positive samples while avoiding collapse into a trivial solution with the use of stop-gradient operators, non-symmetrical siamese networks, regularization loss functions, and others.

In the following subsections, we will explore some pretext tasks that might be relevant to this work.

2.4.1 Self-Prediction

One of the most used self-prediction tasks is masked prediction, which consists in masking parts of the input and making the model predict the content of those masked parts. In an NLP context, we can mask tokens, while in Computer Vision we can mask patches or even individual pixels. BERT [46], for example, is a language representation model that pre-trains a Transformer Encoder using a masked prediction task where 15% of the sentence is masked. Then the masked input is computed by the model that tries to predict the masked tokens using a cross-entropy loss. Denoising autoencoder [52], on the other hand, is a computer vision model that learns in a self-supervised way by denoising a corrupted image. The model is fed with noisy images, i.e. some pixels masked, and tries to learn general representations that are robust against small changes in the input.

2.4.2 Contrastive

The goal of contrastive tasks is to learn such a representation that makes similar samples, also called positive, stay close to each other in representation space while dissimilar ones, also called negatives, are far apart.

2.4.2.A Contrastive Objectives

A key component is the contrastive training objective, which uses the positive and negative samples to adjust the representations of the model. One of the earliest objectives is the Contrastive Loss [53, 54]. Imagine we want to use the loss function to train a function $f_\theta(x_i)$ to learn a good embedding for the input x_i, meaning that samples from the same class have a similar embedding, while samples from different

classes have different embeddings. The loss function, in Equation 2.9, is very straightforward, we have two cases: $y_i = y_j$ and $y_i \neq y_j$. In the first case, the function tries to minimize the distance between the computed representations, in the second case, the function maximizes the distance between the representations lower bounded by a hyperparameter ϵ.

$$\begin{aligned}\mathcal{L}_{cont}(x_i, x_j, \theta) = \mathbb{I}[y_i = y_j] \left\| f_\theta(x_i) - f_\theta(x_j) \right\|_2^2 + \\ \mathbb{I}[y_i \neq y_j] max(0, \epsilon - \left\| f_\theta(x_i) - f_\theta(x_j) \right\|_2)^2\end{aligned} \tag{2.9}$$

A more recent objective, and also currently a popular choice, is the InfoNCE loss [55] which uses a cross-entropy function. In Equation 2.10 we present the loss function as proposed in the original work, where $f_k(x_{t+k}, c_t)$ is given by the Equation 2.11 and z_{t+k} is the result of encoding x_{t+k}. Note that f_k is somewhat similar to a dot product, so we can also see it as a similarity score function between the encoding of the input, z_{t+k}, and a prediction, $\hat{z}_{t+k} = W_k C_t$, similar to the score function in the Self-Attention section. To minimize this function we want to maximize the numerator, i.e. the similarity score of the positive sample to be large, and minimize the denominator, i.e. the similarity score of the negative samples to be small.

$$\mathcal{L}_{CPC} = -\mathbb{E}\left[\log \frac{f_k(x_{t+k}, c_t)}{\sum_{x_j \in X} f_k(x_j, c_t)}\right] \tag{2.10}$$

$$f_k(x_{t+k}, c_t) = \exp(z_{t+k}^\top W_k c_t) \tag{2.11}$$

2.4.2.B SimCLR

SimCLR [50] is a state-of-the-art model that learns visual representations by using a simple contrastive learning framework, Figure 2.9. The two separate networks try to maximize agreement on the respective representations using a contrastive objective, called NT-Xent or normalized temperature-scaled cross-entropy loss, Equation 2.12. The process starts with the sample being augmented by two different operators resulting in different but correlated augmentations, then each augmented sample is computed by a network $f(.)$, in this case, a Residual Network [56] that outputs a representation h. Subsequently, the representations are computed by a simple one hidden layer MLP, $g(.)$, which will result in z. This last function $g(.)$ is only used in training time since it only exists to help the contrastive loss and it is not necessary for generating the real representation of the sample. Unlike the contrastive objectives we have previously seen, the NT-Xent doesn't sample negative samples for each datapoint *per se*. Instead, it samples N datapoints each resulting in $2N$ augmented samples and therefore for each example the other $2(N-1)$ examples will be negatives.

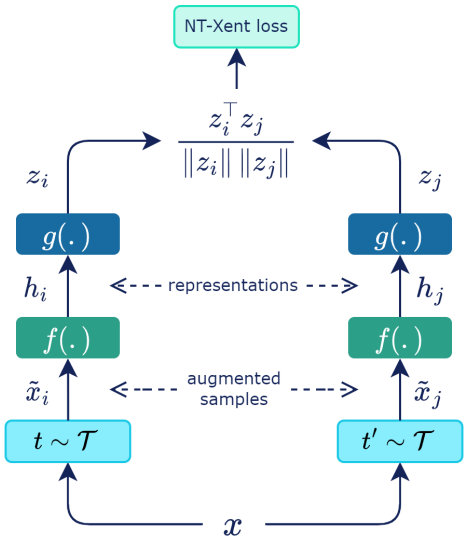


Figure 2.9: SimCLR framework

$$\mathcal{L}_{\text{SimCLR}} = -\log \frac{\exp(\text{sim}(z_i, z_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{I}_{[k \neq i]} \exp(\text{sim}(z_i, z_k)/\tau)} \tag{2.12}$$

2.4.2.C MoCo

Another work in the area of visual representation learning is MoCo [51], which is in many ways very similar to SimCLR but still different enough to be worth talking about, Figure 2.10. The core idea behind MoCo is to have a dynamic dictionary with a queue and a moving-averaged encoder. Like SimCLR we have a siamese network although this time the representations are called a query and keys. MoCo introduces a queue of keys that contains the N previous and the current mini-batch is used, since contrastive methods benefit from large sets of negative samples, in this case, keys. Additionally, it also changes the key's encoder to be a momentum encoder, i.e. the encoder parameters are updating slowly using Equation 2.13, instead of using gradient descent or a simple copy of parameters from the query encoder. The contrastive loss used by the method is an InfoNCE loss, with temperature, of the dot product of the queries with the keys, Equation 2.14.

$$\theta_k \leftarrow m\theta_k + (1-m)\theta_q \tag{2.13}$$

$$\mathcal{L}_q = -\log \frac{\exp(q \cdot k_+/\tau)}{\sum_{i=0}^{K} \exp(q \cdot k_i/\tau)} \tag{2.14}$$

In MoCov2 [57] the authors follow some ideas from the SimCLR and propose adding a simple MLP layer after both encoders to project the representations to a space that is more contrastive loss friendly. For MoCov3 the authors study changing the backbone from a ResNet to a ViT, and introduce changes

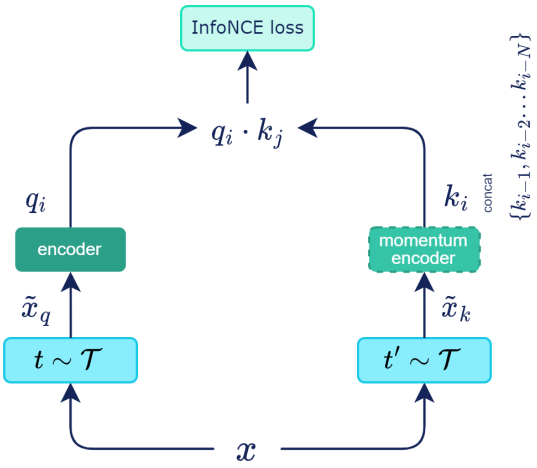


Figure 2.10: MoCo framework

like removing the memory queue, increasing batch size (in part to compensate for the removal of the memory queue), changing the optimizer to a AdamW, adding an extra prediction head, and freezing of ViT's patch projection.

2.4.3 Non-Contrastive methods

2.4.3.A DINO

DINO (self-**di**stillation with **no** labels) [58], like MoCov3, comes from a study of the impact of using self-supervised learning with ViT, however the approach, in this case, is a non-contrastive method based on knowledge distillation [59], as shown in Figure 2.11. This method consists of a siamese network where each encoder is fed with a random transformation of the input. One encoder is called Teacher, more precisely a momentum encoder, and the other encoder is called Student, and their goal is to minimize the cross-entropy between their normalized output probability distributions, computed using a softmax with temperature scaling, given two views of the same source, as shown in Equation 2.15, where $P_t(x)$, and $P_s(x)$, are the probability distributions computed by the teacher and student networks, respectively, x_1 and x_2 are the global views, and V is the set of views. The teacher computation path contains two extra operators when compared to the student, stop-gradient and centering, that contribute to an asymmetry that helps the method avoid collapse. The centering operation consists in adding a bias c to each representation computed by the teacher, where c is a moving average of the mean over the batch of representations computed by the teacher, as shown in Equation 2.16. Unlike, most methods, DINO creates more than 2 augmentations of the same source, more precisely it creates a set of views composed of two global views and several local views. All views are computed by the student network while only the global views are computed by the teacher network, which pushes the student to create a

local-to-global correspondence.

$$\mathcal{L} = min \sum_{x \in \{x_1, x_2\}} \sum_{\substack{x' \in V \\ x' \neq x}} H(P_t(x), P_s(x')) \tag{2.15}$$

$$c \leftarrow mc + (1-m)\frac{1}{B}\sum_{i=1}^{B} y_t^i \tag{2.16}$$

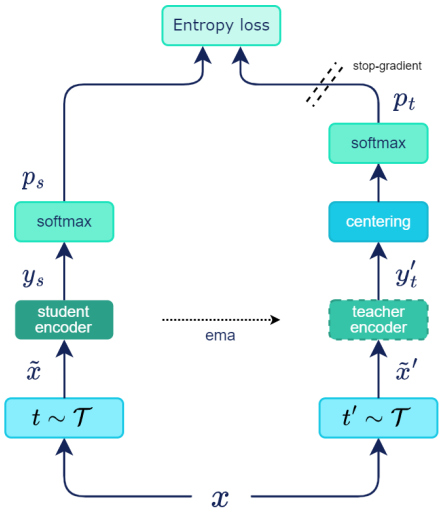


Figure 2.11: Simplified diagram of DINO's architecture

3

Related Work

Contents

3.1 Pretraining representations

Previous work, similarly to our approach, has explored pretraining representations using self-supervised methods which led to great data-efficiency improvements in the RL phase [8, 60] or superior results in evaluation tasks, like AtariARI [61]. Others have pretrained representations using RL algorithms, like DQN, and transfer those learned representations to a new learning task [62].

PIE-G (Pretrained Image Encoder for Generalizable) [63] uses an encoder pre-trained on ImageNet to improve generalization in environments where the backgrounds are changing.

3.2 Temporal Relations

Other works have explored learning representations that have temporal information encoded. ATC (Augmented Temporal Contrast) [9] trains an encoder to compute temporally consistent representations using contrastive learning, and the ST-DIM (SpatioTemporal DeepInfoMax) [61] captures spatial-temporal information by maximizing the mutual information between features of two consecutive observations.

3.3 Joint learning

In recent years, adding an auxiliary loss to the RL loss, usually called joint learning, has become a common approach by many proposed methods. Curl [64] adds a contrastive loss using a siamese network with a momentum encoder. Another work studies different joint-learning frameworks using different self-supervised methods [65]. SPR [66] uses an auxiliary task that consists of training the encoder followed by an RNN to predict the encoder representation k steps into the future. PSEs [67] combines a policy similarity metric (PSM), that measures the similarity of states in terms of the behaviour of the policy in those states, and a contrastive task for the embeddings (CME) that helps to learn more robust representations. PBL [10] learns representations through an interdependence between an encoder, that is trained to be informative about the history that led to that observation, and an RNN that is trained to predict the representations of future observations. Proto-RL [68] uses an auxiliary self-supervised objective to learn representations and prototypes [69], and uses the learned prototypes to compute intrinsic rewards which will push the agent to explore the environment.

3.4 Augmentations

While we only use augmentations in the pre-training phase, their use during reinforcement learning has also been studied. Methods like DrQ [70] and RAD [71] pair an RL algorithm, like SAC, with image

augmentations to improve data efficiency and generalization of the algorithms. Their use is not trivial and both methods explore the impact of the different augmentations commonly used in computer vision tasks. RAD, for example, that different games might benefit more from different augmentations, but in general ”random translation” and ”random crop” have the highest impact.

3.5 Vision Transformer for vision-based Deep RL

Recent works, also compare the Vision Transformer to convolution-based architectures with a similar number of parameters and show that ViT is very data inefficient even when paired with an auxiliary task [72].

3.6 Self-Supervised Learning from image sequences

Previous works also explore learning temporally coherent representations through a diverse set of approaches. Shuffle-and-Learn [14] introduces an unsupervised temporal order verification task, which we use for this work, and that was originally used to train a model to predict human poses in videos. VITO [73] pretrains an encoder over image sequences instead of images by approximating a source t and a source $t + k$ where k is sampled using a normal distribution with center at zero. Transporter [74] learns to transform source frames from Atari games into another, temporally distant, frame which allows the model to learn the important features present in the frame. For object recognition in videos, it has explored the use of a ranking loss function, which is similar to a contrastive loss and that trains the encoder to be invariant to changes in the patch being tracked [75]. Another previous work proposes an approach that uses a low-level motion-based grouping to pretrain an encoder in an unsupervised way for object segmentation in video [76].

Multiple works not related with RL also propose simple pretext tasks to train encoders to capture information from image sequences. These pretexts tasks can be playback speed classification [77], a temporal order classification [14, 78, 79], a jigsaw game [80] or a masked modelling task [81]. A different approach consists of using contrastive learning. In this category, we can find methods that maximise the similarity between image sequences [82], use autoregressive models to predict frames multiple steps in the future [83], and maximize the similarity between temporally adjacent frames [84].

3.7 VICReg

VICReg [13] is a non-contrastive method that trains a network to be invariant to augmentations applied to the inputs while avoiding a trivial solution using losses that act as regularizers over the embeddings.

While VICReg is agnostic concerning the architectures used and even the weight sharing, in this work we consider the version where paths are symmetric, the weights are shared, and each path is composed of an encoder (also called backbone) and an expander, as shown in Figure 3.1. In addition, the expander also removes information that is not common to both representations. Besides the invariance loss, which trains the model to be invariant to augmentations by minimizing the L2 loss of the embeddings, z and z', from the two computation paths, two additional loss functions are used, called variance and covariance. The variance loss is a hinge loss that pushes the variance of the variables from the embedding to be above a certain threshold. The covariance loss regularizes the embeddings by minimising the sum of the squared off-diagonal coefficients of the embedding covariance matrix and consequently decorrelating the variables of the embedding. Furthermore, the expander increases the dimension of the representation vector in a non-linear way allowing the covariance loss to reduce dependencies and not only correlations of the representation vector. Thanks to its regularization losses, which ensure that the final method won't suffer from any type of collapse, VICReg offers a simple and non-contrastive framework that can be easily extended.

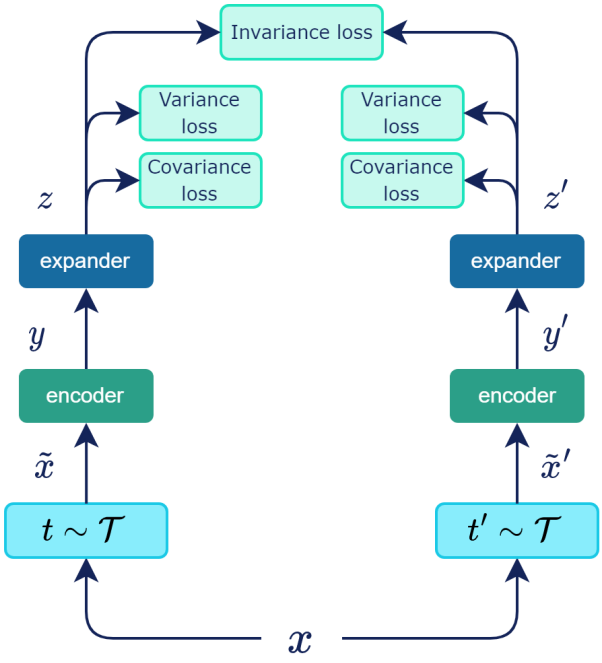


Figure 3.1: VICReg architecture

4

Self-Supervised Pre-Training

Contents

4.1 TOV-VICReg

VICReg uses three loss functions: **invariance** is the mean of square distance between each pair of embeddings from the same original image, as shown in Equation 4.1, where Z, and Z' are two sets of embeddings, of size N, that result from computing two different augmentations of N sources, and z_j denotes the *j-th* embedding in the set; **variance** is a hinge loss that computes, over the batch, the standard deviation of the variables in the embedding vector and pushes that value to be above a certain threshold, as shown in Equation 4.2, where d denotes the number of dimensions of the embedding vector, and Z^j is the set of the *j-th* variables in the set of embedding Z; **covariance** is a function that computes the sum of the squared off-diagonal coefficients of a covariance matrix computed over a batch of embeddings, as shown in Equation 4.3, to decorrelate the variables from the embedding. While the invariance loss function tries to make the model invariant to augmentations, i.e. output the same representation vector, the other two functions regularize the method by pushing the variables of the embedding vector to vary above a certain threshold and decorrelating the variables in each embedding vector.

$$i(Z, Z') = \frac{1}{N} \sum_{j}^{N} \|z_j - z'_j\|_2^2 \tag{4.1}$$

$$v(Z) = \frac{1}{d} \sum_{j}^{d} \text{max}(0, \gamma - \sqrt{Var(Z^j)}) \tag{4.2}$$

$$c(Z) = \frac{1}{d} \sum_{i \neq j} [\text{Cov(Z)}]_{i,j}^2 \tag{4.3}$$

TOV-VICReg or Temporal-Order-Verification-VICReg extends VICReg to better capture the temporal relations between consecutive observations and consequently encode extra information that can be useful in the deep reinforcement learning phase. To achieve that we add a new temporal order verification task, as seen in Shuffle-and-Learn [14], that consists of a binary classification task where a linear layer learns to predict if three given representation vectors are in the correct order or not. Like the other losses, we also employ a coefficient for the temporal loss. Figure 4.1 visually illustrates TOV-VICReg.

At each step we sample 3 consecutive observations, $\{x_{t-1},\ x_t,\ x_{t+1}\}$, x_t is processed by two different augmentations, and like VICReg these are the augmentations used in BYOL [85], while x_{t-1} and x_{t+1} are processed by two simple augmentations composed of a color jitter and a random grayscale, as shown in the pseudocode available at Appendix A.8. The x_t augmentations are computed by the VICReg computation path and the resultant embeddings are used for the loss functions, i.e. variance, invariance, and covariance. In the **temporal order verification task** we encode the augmentation of x_{t-1} and x_{t+1}, and concatenate those two representations with one of the representations of x_t, in our

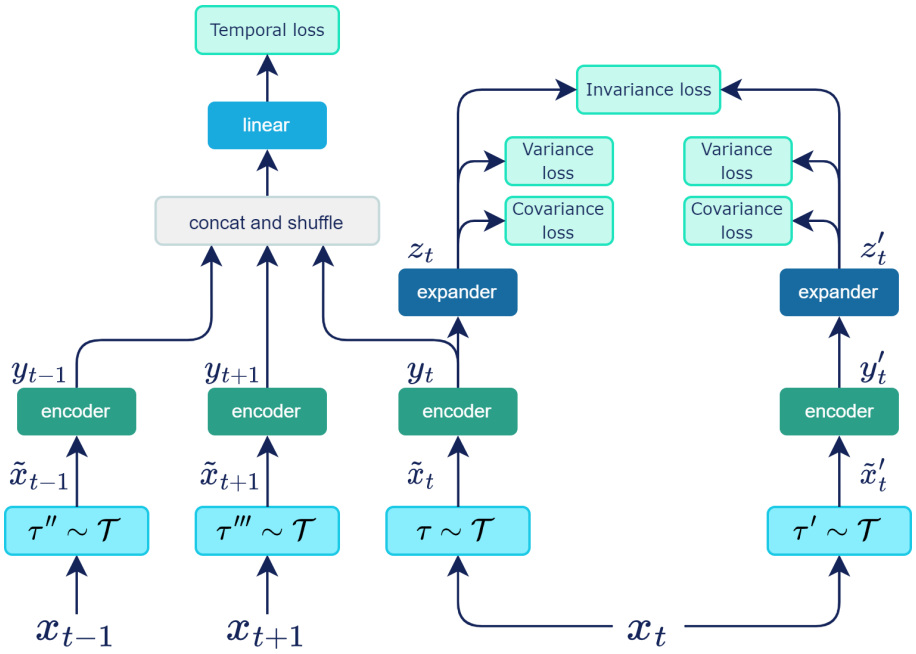


Figure 4.1: TOV-VICReg architecture

case we used the one that was augmented without solarize, obtaining the vector $\{y_{t-1}, y_t, y_{t+1}\}$. At last, we randomly permute the order of the representations in the vector and feed the resultant concatenated vector to a linear layer with a single output node that predicts if the given concatenated vector has the representations in the right order or not. The **temporal loss** used to optimize the model for this task is a Binary Cross Entropy loss, as shown in Equation 4.4. Lastly, the complete loss used to optimise TOV-VICReg is presented in Equation 4.5. TOV-VICReg's pseudocode can be found in Appendix A.7.

$$t(\hat{r}, r) = \frac{1}{N} \sum_{j}^{N} -(r \log \hat{r} + (1 - r) \log (1 - \hat{r})) \tag{4.4}$$

$$\mathcal{L}(Z, Z', r, \hat{r}) = \text{inv\_coef} \times i(Z, Z') + \text{var\_coef} \times (v(Z) + v(Z')) + \text{cov\_coef} \times (c(Z) + c(Z')) + \text{temp\_coef} \times t(\hat{r}, r) \tag{4.5}$$

4.1.1 Loss coefficients

As we stated all four objectives are added into a singular weighted sum. However, choosing different values for these coefficients can lead to very different results. So to start we used VICReg's default values for the respective losses and picked a similar value for the temporal loss, as shown in Table 4.1. However, these values caused a lot of instability which would result in losses exploding. To solve

this problem we decreased the temporal coefficient until we got a stable pretraining, which resulted in this value changing to 0.1. When we evaluated the representations using the metrics presented in Section 6.1 we also observed that the correlation coefficient was higher than expected (~ 0.15). So, we tested increasing this value to reduce correlations and dependencies between the variables in the representation vector and arrived at the value of 10.0. Where we found no improvement in the correlation coefficient when using higher values for this loss coefficient. The final coefficients can be found in Table A.7.

Loss	Coefficient
Invariance	25.0
Variance	25.0
Covariance	1.0
Temporal	25.0

Table 4.1: Original coefficients for each loss

4.2 Pre-Training Methodology

We pretrained four encoders, one using our proposed method TOV-VICReg and three using state-of-the-art self-supervised methods: MoCov3 [86], DINO [58] and VICReg [13]. For this study, the encoder used is a Vision Transformer, more precisely the ViT tiny with a patch size of 8. We chose this patch size based on experiments that show that this value performed well in terms of data-efficiency when compared to 6, 10, and 12 without being too computationally intensive (Appendix A.1). Moreover, the implementation we use is an adaptation of the timm library [87] implementation, which can be found in the source code of the DINO method. The dataset used is a set of observations from 10 of the 26 games in the Atari 100k benchmark, all available in the DQN Replay Dataset [88]. For each game, we use three checkpoints with a size of one hundred thousand data points (observations), which makes up a total of three million data points (55 hours). The pretraining phase is 10 epochs with two warmup epochs. We used the official code bases of all the self-supervised methods and tried to change the least amount of hyperparameters. Appendix A.9 contains the tables with the hyperparameters used for each method.

4.3 Alternative methods explored

Beyond the temporal order verification task we also explored several different pretext tasks that also tried to distil some temporal information into the encoder.

- In our first approach we tried a spatio-temporal loss, with the goal of making observations close in time also close in space. The loss is a hinge function that minimizes the L2 distance until a

certain threshold, as shown in Equation 4.6. Looking back we think we could have improved it more by considering more than consecutive frames ($t+k$ and $t-k$, where $k=1$) and instead, for example, sample a value for k in a Gaussian distribution. However, the temporal order verification task ended up proving to be very effective and we didn't further explore this approach.

- The third approach we explored was a trivial continuation of the temporal order verification task, which consisted in changing the task from a binary to a multi-class classification problem. In this case, the temporal loss was a Cross entropy loss instead. This task proved to be too difficult for the network to solve, where the loss value stayed constant for the entire pretraining.

$$\mathcal{L}(Z, Z') = \frac{1}{N}\sum_{j}^{N} \max(0, \nu - \left\|z_j - z'_j\right\|_2^2) \tag{4.6}$$

5

Data-Efficiency in Reinforcement Learning

Contents

5.1 Training Methodology

In this section, we study the pretrained Vision Transformers, from the previous section, in the context of data-efficiency regime in DRL. And compare them against a randomly initialized ViT tiny and two convolution based networks, the Nature CNN [89], and a ResNet with an amount of parameters similar to ViT tiny (Appendix A.3) that has a size roughly similar to the ViT tiny. To achieve this we took the Rainbow network, replaced the representation module, in this case, a CNN, with different encoders and trained the agent in an Atari game for 100 thousand steps. Figure 5.1 depicts the complete process of pretraining an encoder model (in our case a Vision Transformer) using a self-supervised method and using it in the network employed by the Rainbow algorithm.

To study the data-efficiency of the algorithm Simple [90] in Atari games the authors proposed a new benchmark, called Atari 100k, which consists in training an agent with a smaller amount of training data. Instead of the typical 25 or 50 million environment steps, the agents are trained with only 100 thousand steps (with a frame skip of 4, meaning that in reality, it's 400 thousand steps). Several works [8, 20, 64, 66, 70, 71, 91–93] have followed this approach to test data-efficiency in Atari games and we consider it as well for this work.

We trained our agents using a PyTorch implementation of the Rainbow algorithm available on GitHub [94], which offers enough flexibility to adapt it to our needs. In Table 5.1 we present a comparison between the implementation used and the official results reported by DER [92] and Table A.10 contains the hyperparameters used. We observed a similar performance in most games except for Assault, and Frostbite, where the official results are significantly higher. Despite these differences, we validated the implementation code and are confident that the results here presented are trustworthy. To allow the agents to play the Atari games we used the gym library [95], where for all games we used the 4th version without frame skip, e.g. "AlienNoFrameskip-v4", and wrapped it with the DQN wrappers. Appendix A.5 contains the code used to wrap the environments in Gym.

Game	DER	DER (ours)
Alien	739.9	446.6 ± 224.7
Assault	431.2	178.7 ± 87.1
Bank Heist	51.0	23.8 ± 14.3
Breakout	1.9	1.93 ± 1.43
Chopper Command	861.8	696.0 ± 274.6
Freeway	27.9	27.8 ± 2.0
Frostbite	866.8	127.7 ± 25.8
Kangaroo	779.3	448.0 ± 648.0
MsPacman	1204.1	1015 ± 487.1
Pong	-19.3	-18.6 ± 4.4

Table 5.1: Comparison between DER scores and our implementation scores

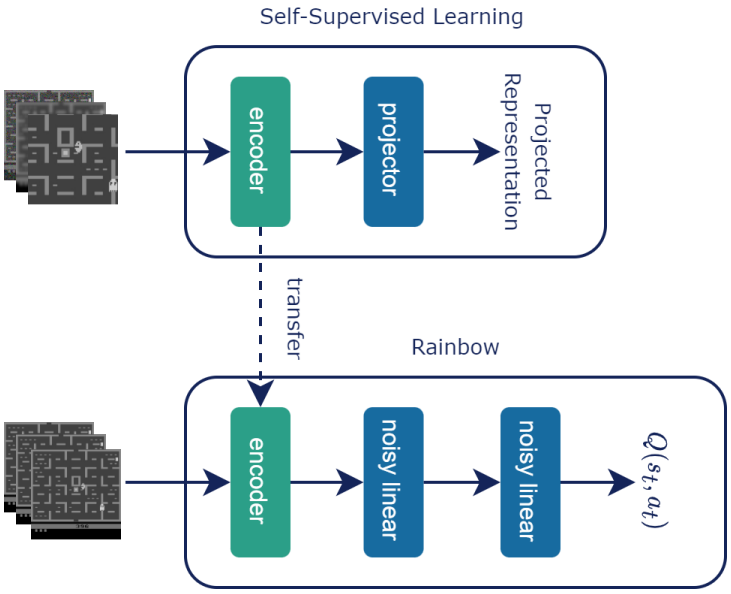


Figure 5.1: Simplified diagram of the model transfer

5.2 Results Methodology

Results in Reinforcement Learning suffer from a lot of uncertainty, especially in settings like the Atari 100k. To reduce this uncertainty one can perform more training runs across different seeds but this is computationally expensive. In this work, we present our results using the RLiable framework [91]. This consists in using stratified bootstrap to sample from the set of normalized scores obtained for each game. Meaning that all games are proportionally represented in the set of sampled scores, which is then used to compute the different metrics and the 95% confidence intervals.

5.3 Results

Figure 5.2 shows the aggregate metrics on 10 Atari games with training runs of 100k steps. Starting with the non-pretrained models (ViT, Nature CNN, and SGI-ResNet Large) we can assess that, observing the mean, Nature CNN is the most sample efficient model followed by SGI-ResNet Large, and ViT, respectively. Regarding the pretrained models, ViT, when pretrained with our method, performs better than the other models and the non-pretrained ViT in all metrics. It is worth noting that we report a higher variance in the results of our proposed method when compared to the remaining methods and non-pretrained models. ViT+TOV-VICReg when compared to Nature CNN, which has far fewer parameters, and SGI ResNet Large, with a similar number of parameters seems to closely match their sample-efficiency performance (Appendix A.3). Furthermore, the difference between the non-pretrained ViT and ViT pretrained with TOV-VICReg shows that a good self-supervised method that explores temporal relations

and 3 million data points can help close the sample-efficiency gap while remaining a more complex and capable model. Regarding the remaining self-supervised methods, MoCo seems to perform considerably well obtaining even a median very similar to TOV-VICReg and is then followed by DINO and VICReg, respectively. All pretrained ViTs show an improvement in comparison to the non-pretrained ViT.

$$\frac{\text{score} - \text{random\_score}}{\text{average\_human\_score} - \text{random\_score}} \tag{5.1}$$

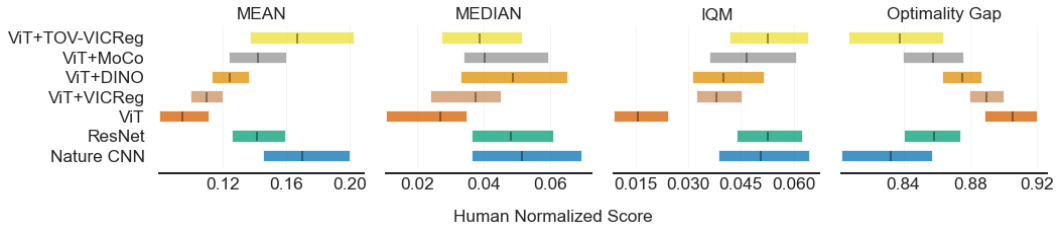


Figure 5.2: The eval runs across the different games are normalized and treated as a single task. The IQM corresponds to the Inter-Quartile Mean among all the runs, where the top and bottom 25% are discarded and the mean is calculated over the remaining 50%. The Optimality Gap refers to the number of runs that fail to surpass the human average score, i.e. 1.0.

5.3.1 Unseen environments

Table 5.2 shows a comparison of the non-pretrained and pre-trained (using TOV-VICReg) Vision Transformer in Atari games that were not used in the pre-training phase. In general, both models seem to perform very similarly as indicated by the Inter-Quartile-Mean (IQM) over the aggregated normalized scores, except for two games, RoadRunner where the pretraining seems to degrade data-efficiency and Venture where pretraining improves data-efficiency. In short, we don't find any advantage in using a pre-trained vision transformer for games that were not used during pretraining. We don't find this result surprising given the lack of variety present in the dataset used for pretraining which reduces the possibility of the encoder finding features that can be used elsewhere.

Games	ViT	TOV-VICReg+ViT
Asterix	443.5 ± 225.6	445.0 ± 214.9
Krull	944.5 ± 525.8	708.9 ± 572.3
RoadRunner	2687.0 ± 2884.3	913.0 ± 1289.9
SpaceInvaders	184.3 ± 117.0	155.9 ± 91.0
Venture	4.0 ± 28.0	76.0 ± 152.4
IQM	0.0174	0.0186

Table 5.2: Mean and standard error results of the evaluations across 10 different training runs, where at each evaluation the agent plays 10 episodes of the game. The agent was trained using the Rainbow algorithm for 100k steps.

6

Evaluate Representations

Contents

In this chapter, we explore some metrics and visualizations that can help us compare the different pretrained encoders. Some are more objective such as the metrics and cosine similarity while others, like the attention maps and t-SNE, are more subjective and are intended to give an intuition of their properties. Together they can help us explain the results in the previous chapter. We also propose using an evaluation task which consists of imitation learning using a frozen encoder and a linear layer (also called linear probing) for the action prediction that helps evaluate the pretrained encoders more efficiently and with less uncertainty.

6.1 Metrics

A significant phenomenon when doing self-supervised training is the collapse of the representations, which can be seen in three forms: representational collapse, dimensional collapse, and informational collapse. Representational collapse refers to the features of the representation vector collapsing to a single value for every input, meaning the variance of the features is zero, or close to zero. In dimensional collapse, the representations don't use the full representation space, which can be measured by calculating the singular values of the covariance matrix calculated over the representations. Informational collapse defines the case where the features of the representation vector are correlated and therefore are representing the same information.

6.1.1 Dimensional Collapse

To compute the dimensional collapse metric we first calculate the covariance matrix of the representations generated by the different encoders we are exploring hover a batch of observations from all the 10 games we used for the pretraining. Then we compute the singular of the resulting matrix using singular value decomposition and plot them ranked by value.

All methods seem to avoid dimensional collapse, i.e. most dimensions have a singular value larger than zero, as observed in Figure 6.1. However, we notice that some methods make better use of the space available since they present higher singular values. TOV-VICReg, in particular, seems to excel in this metric, even improving the results obtained by VICReg. It is worth noting that both VICReg and TOV-VICReg employ a covariance loss that helps decorrelate the embedding variables which may be contributing positively to these results. Furthermore, we used a covariance coefficient of 10 for TOV-VICReg and 1 for VICReg, a change that according to our experiments culminates in the increase here observed.

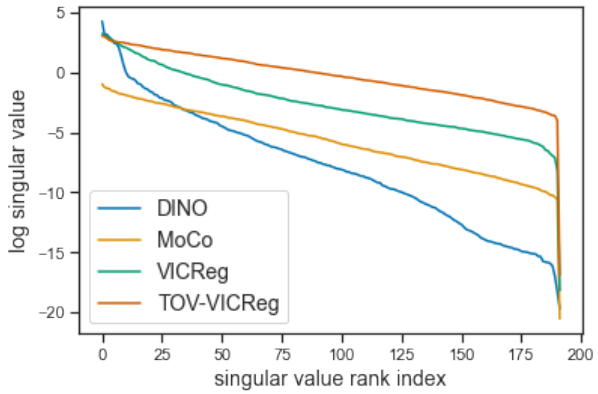


Figure 6.1: Logarithm of the singular values of the representation vector's covariance matrix sorted by value.

6.1.2 Representational Collapse

Results in Table 6.1 show the computed standard deviation of the representation vector over a batch of thousands of data points. DINO, VICReg and TOV-VICReg show a value well above zero, meaning that none of the methods suffered from representation collapse during training. On the other hand, MoCo shows a much smaller value of 0.178, which is still far from a complete collapse. Both VICReg and TOV-VICReg use a hinge loss that pushes the representation vector to have a standard deviation of 1 or above. While VICReg slowly converges to this value our method converges to roughly 1.65, which might be the result of adding a temporal order verification task.

DINO	MoCo	VICReg	TOV-VICReg
0.979	0.178	1.003	1.648

Table 6.1: Average standard deviation of the representation vector

6.1.3 Informational Collapse

We report in Table 6.2, the comparison of the average correlation coefficients of the representation vectors. TOV-VICReg performs better than the other methods, including VICReg, which present very similar coefficients. Like in the dimensional collapse, this result is in part due to the higher covariance coefficient used in TOV-VICReg which by design helps the model to decorrelate the representation's features. Increasing the coefficient in VICReg results in a lower correlation coefficient as well, but is still higher than TOV-VICReg.

DINO	MoCo	VICReg	TOV-VICReg
0.1764	0.1538	0.1531	0.0780

Table 6.2: Average correlation coefficient

6.2 Visualizations

In this section, we present different visualizations to better understand the representations learned by each of the methods. Our goal with the following visualizations is to help us better understand the learned representations and give some intuitions about their properties.

6.2.1 Cosine similarity

Figure 6.2 presents a similarity matrix of the representations where we can observe that TOV-VICReg can better distinguish between observations of different games but also observations from the same game, as shown in Figure 6.3. MoCo, on the other hand, seems to make a good distinction between observations from the same game. However, we can observe in the colour bar that all the representations are very similar to each other, which corroborates the results obtained in Section 6.1. Oppositely, VICReg and DINO manage to spread representations more, as we can see in the colour bars, but the yellow squares in the diagonal show that the representations from the same game are more similar to each other which is corroborated by Figure 6.3. Given the empirical results, we believe that this capacity to distinguish observations from the same game might be a good indicator.

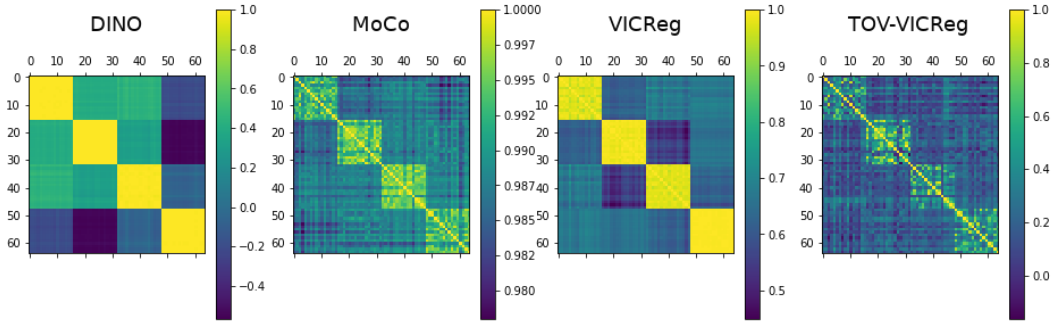


Figure 6.2: Similarity matrices of the representations computed by MoCo, DINO, VICReg, and TOV-VICReg respectively. There are a total of 64 data points, from 4 different games: Alien, Breakout, MsPacman, and Pong, where from 0-15 are from Alien, 16-31 are from Breakout and so forth.

6.2.2 Attention maps

The research work that proposes DINO shows that the Vision Transformer is able to attend to important parts of the input after training using DINO. Inspired by these results, we try to make the same evaluation

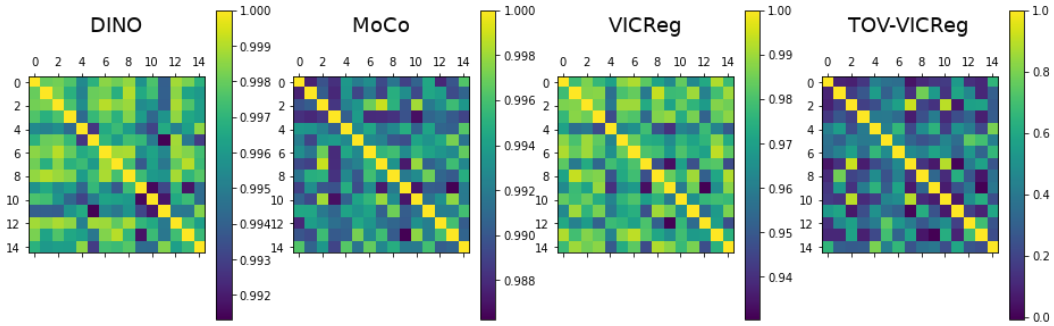


Figure 6.3: Similarity matrices of the representations computed by MoCo, DINO, VICReg, and TOV-VICReg respectively, of observations from MsPacman.

for the several self-supervised methods we are studying, including TOV-VICReg, and try to understand if any of the encoders can attend to interesting parts of the input. In Figure 6.4, we can see the results of all methods for an observation from the game of Pong, where each method produces three attention maps, one for each self-attention head of the last block of the Vision Transformer. All pretrained ViT seem to attend at some level to important game features like the ball and the paddles. However, TOV-VICReg is the only method that doesn't spread the attention to other parts of the frame that we don't consider important to describe the current state of the game. When comparing to VICReg's attention maps we believe that the temporal order verification task greatly helped the attention of the model. In more visually complex games, e.g. Freeway or MsPacman, these attention maps start to be more difficult to analyse but it is still possible to discern some important features.

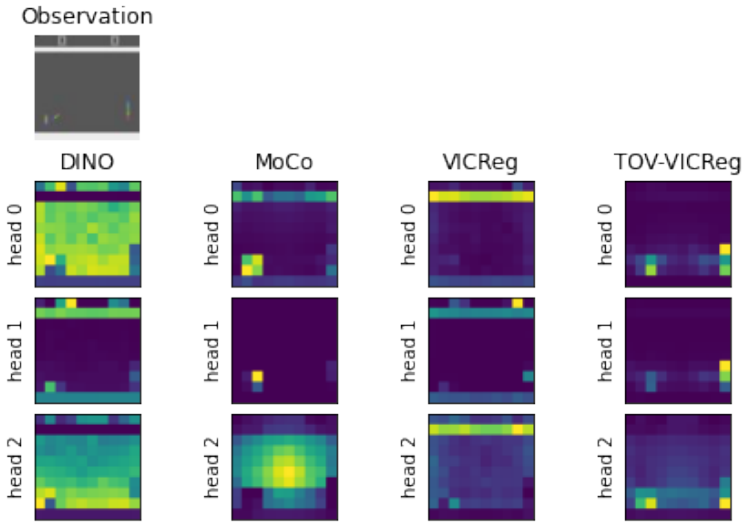


Figure 6.4: Attention maps for Pong
Attention maps produced by the pretrained ViTs. We fed a pretrained ViT with an observation from the game Pong and obtained the attention maps from the three heads in the last block.

6.2.3 t-SNE

A commonly used method to visualise representations generated by networks in 2D is the t-SNE algorithm [96]. t-SNE is a tool that can reveal interesting phenomena that might hint into what is happening when we train the Vision Transformer using different self-supervised learning methods.

In Figure 6.5, we observe that some structure seems to emerge at the embeddings generated by the t-SNE algorithm from the TOV-VICReg and MoCo representations. In both it is possible to discern a path of points with the same colour, meaning that the correspondent observations are close in time and are also close in the representation space. On the other hand, DINO appears to present clusters of points with similar colours but are more dispersed when compared to MoCo and TOV-VICReg. VICReg's t-SNE shows some clustering of points with similar colours but are less well defined than the others.

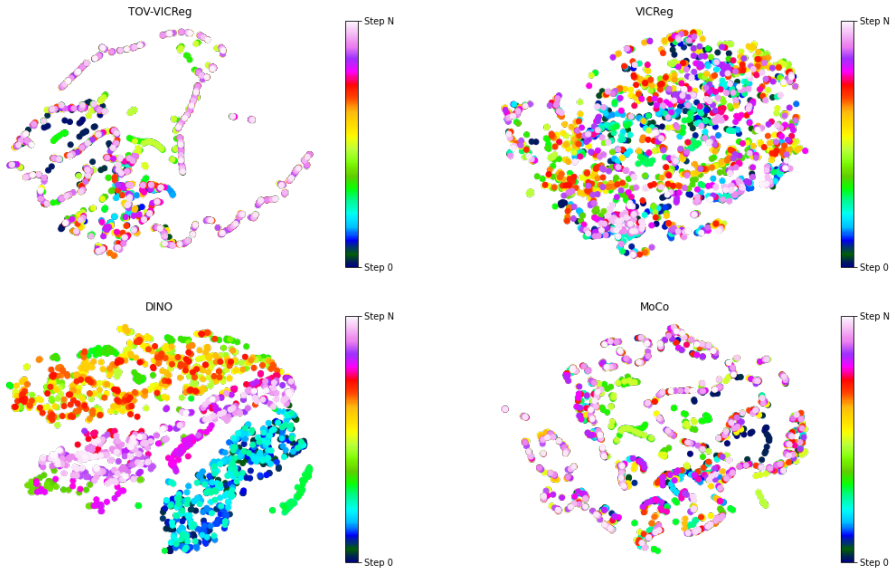


Figure 6.5: t-SNE of the representation space generated by Vision Transformer pretrained with different self-supervised methods. All points correspond to observations from the game Pong.

6.3 Evaluation Task

Evaluating representations computed by a pretrained encoder is a difficult task. One possible option is assessing improvements in data efficiency in a reinforcement learning task, as we did in the previous section. However, the results usually suffer from a high level of uncertainty which requires us to run dozens of training runs, thus making it computationally expensive. Another possible path would be using previously proposed benchmarks like the AtariARI benchmark [61], which tries to evaluate representations using the RAM states as ground truth labels. However, this only works for 22 Atari games (out of 62) and requires the encoder to use the full observation provided by the environments (160x210).

		Randomly initialized encoder			Pre-trained encoder					W/o freeze
Game	Random Classifier	Nature CNN	ResNet	ViT	ViT+TOV-VICReg	ViT+DINO	ViT+MoCo	ViT+VICReg	ViT+TOV-VICReg L	Nature CNN
Alien	0.0556	0.0077	0.0558	0.0147	0.1003	0.0470	0.0646	0.0695	0.0988	**0.1021**
Assault	0.1519	0.1497	0.2270	0.1770	0.3044	0.2536	0.2557	0.3704	0.3065	**0.6673**
BankHeist	0.0608	0.0780	0.1312	0.0756	0.1622	0.1059	0.1083	0.1467	0.1523	**0.2080**
Breakout	0.2509	0.1311	0.3850	0.2183	0.3285	0.3591	0.2765	0.4077	0.3099	**0.5907**
Chopper Command	0.0563	0.0145	0.0647	0.0176	**0.3225**	0.0383	0.2019	0.1298	0.3088	0.2660
Freeway	0.3999	0.6808	0.6850	0.6843	0.7041	0.6850	0.6972	0.6971	0.6942	**0.8885**
Frostbite	0.0565	0.0302	0.0730	0.0367	**0.1021**	0.0517	0.0744	0.0664	0.1001	0.1019
Kangaroo	0.0603	0.0311	0.1039	0.0562	0.2184	0.0877	0.1374	0.1259	0.2126	**0.3311**
MsPacman	0.1121	0.0388	0.1419	0.0780	0.1527	0.1215	0.1168	0.1400	0.1500	**0.2063**
Pong	0.1644	0.0692	0.1702	0.0718	0.2853	0.1447	0.2730	0.2337	0.3042	**0.4340**
Mean	0.1369	0.1231	0.2038	0.1430	0.2680	0.1894	0.2206	0.2387	0.2637	**0.3796**

Table 6.3: F1-scores for each game evaluated and mean. We trained all the encoders in all games seperatly for 100 epochs over a dataset of 100k observations and evaluate in 10k new observations. The rightmost column show the results of a Nature CNN encoder that was not frozen during train and which we use as a goal for the remaining.

For those reasons, we propose using a different evaluation task that is more efficient, allowing us to test more pretrained models during the research process (50min per game), and flexible, meaning that we can use it in different environments. Our evaluation task is a simple Imitation Learning task where we train a network, composed of a frozen pre-trained encoder and a linear layer, i.e. linear probing, to correctly predict the action that a certain policy will perform given its current observation. The intuition to use such an evaluation is that a representation that allows an agent to efficiently learn an environment must encode state information that can be recovered by a linear layer and which can be used to learn other tasks efficiently.

We present the results in Table 6.3, we compare against a random classifier, i.e. uniform sampling, randomly initialized networks and a non-frozen encoder which we use as a goal score. All methods were trained for 100 epochs except the latter which we trained for 300. We use the DQN Replay dataset to obtain the observations and the actions we obtain the datapoints from the last checkpoint of each game, where we consider the policy to be less stochastic. The train dataset is composed of 100 thousand observations from the game we are testing and the test dataset is composed of 10 thousand. ViT+TOV-VICReg L corresponds to a ViT tiny pretrained with TOV-VICReg on the 26 Atari games from the Atari100k.

To validate our evaluation task we calculate the Pearson correlation coefficient between the mean of the average human normalized scores, obtained in the Reinforcement Learning, and the mean of the F1-scores, from the evaluation task of all pretrained models. We report a Pearson correlation factor of 0.6985. Even though we are not in the presence of a strong correlation there is a clear trend for the RL scores to increase when the evaluation scores also increase, as observed in Figure 6.6. Despite the promising results, more data points are needed, especially using different pre-training methods, which would allow us to better validate this evaluation task. Nevertheless, we believe that the evaluation task might be a compelling tool for future methods that try to learn good representations for a Reinforcement Learning task.

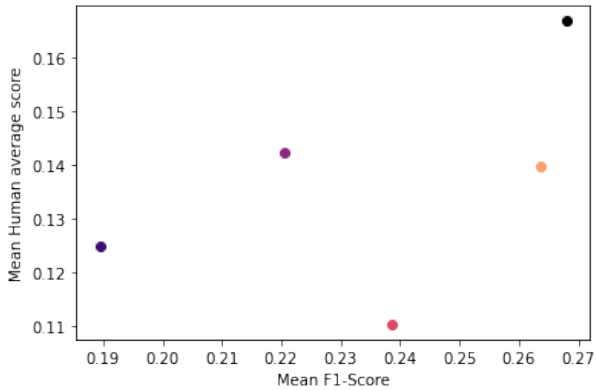


Figure 6.6: Relation between the mean average human score obtained in RL and the mean F1-score obtained in the evaluation task of several experiments

7

Generalization

Contents

Generalization is used to refer to the capability of a model to maintain its performance when used in previously unseen data. In this work, we evaluate generalization using the procedurally generated environments available at procgen [15], which contains several arcade games each with an almost infinite number of distinct levels. During training, the agent only has access to a subset of the available levels and at the end of the training, we test the trained agent in ten thousand episodes sampled from previously unseen levels. With this approach, we can understand if the agent is learning generalizable features and behaviours by measuring the gap between the train and test run.

7.1 Evaluating generalization

For this section, we will follow the experimental protocol proposed for procgen [15] where we train an agent using the PPO algorithm for 25 million steps at difficulty "easy" and with only 200 levels available (seed 0-199). As in Chapter 5 we will study different models for the representation module of the network. However, in this study we will only consider three models: Impala ResNet, Vision Transformer, and Vision Transformer pretrained using TOV-VICReg.

We use the PPO implementation available in the CleanRL library [97]. A benchmark of the implementation in several procgen games can be found in Appendix A.13 and the hyperparameters used in Table A.11.

7.2 Dataset generation

For the Atari games we took advantage of the DQN Replay Dataset. However, to the best of our knowledge, there is no dataset for procgen games with the same quality and publicly available. For this reason, we created our dataset, which is composed of observations from 9 different games: BigFish, BossFight, Chaser, Climber, CoinRun, Dodgeball, Leaper, Maze, and Miner. To obtain the observations we trained an agent using the PPO algorithm with the Impala ResNet for 10 million timesteps. Each environment set to easy and the agent only played in the levels from 0 to 199, the same levels where we train the agents when evaluating generalisability. Like, in the DQN Replay Dataset we separate the dataset into checkpoints, with 1 million steps each.

We follow a similar training methodology as in Section 4.2. Our final dataset used to train Vision Transformer (ViT) using TOV-VICReg is composed of the first 100k steps of the checkpoints 0, 5, and 9 from the nine games mentioned above.

7.3 Results

For this experiment, we compare the pretrained ViT tiny against the non-pretrained ViT tiny and the Impala ResNet [98] in four different games: Maze, Miner, CoinRun, and Climber. Figure 7.1, shows the average normalized learning curve and average normalized testing score. As we assessed in Chapter 5 the pretrained ViT presents much better data-efficiency. However, this time the model is far behind the convolution-based network. In terms of generalization, the results show a larger gap between the train and test scores corresponding to TOV-VICReg+ViT than the ViT scores gap, even though the test score is higher. This result contrasts with previous works which show generalization improvements when using pretrained encoders [63, 67]. We hypothesise that the lack of diversity present in the dataset used during pretraining can be a limitation in our approach since it doesn't allow the encoder to be more robust to unseen data. Despite this, when we analyze the scores for each game, Figure 7.2 we also find that the test score in two games (Maze, Climber) is lower despite having a higher final training score. This result contrasts with previous work that showed improvements in generalization from agents using pretrained encoders [63]. Note that in the case of procgen it's not possible to play a single level while changing different sprites, like the background, in between episodes. A feature that would allow us to evaluate only the generalization capabilities of the encoder. For that reason, the gaps observed in the plots are partially the result of poor generalization of behaviour. Nevertheless, the observed test score degradation shows more than an inability to generalize behaviours and for that reason, the problem must be in part due to poor representations.

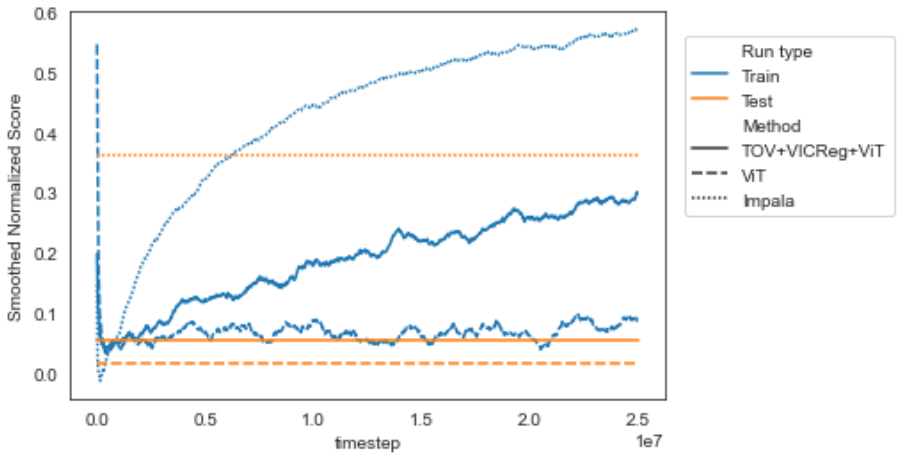


Figure 7.1: Average normalized learning curves and average normalized test scores for all games

To better understand these results we further analyse them using the games Climber and Miner, where we observed a degradation and an improvement, respectively, of the test score. The games are visually different, as shown in Figure 7.3, and also change visually in different ways.

Miner, for example, has a static background at the entire level which in the beginning is obstructed

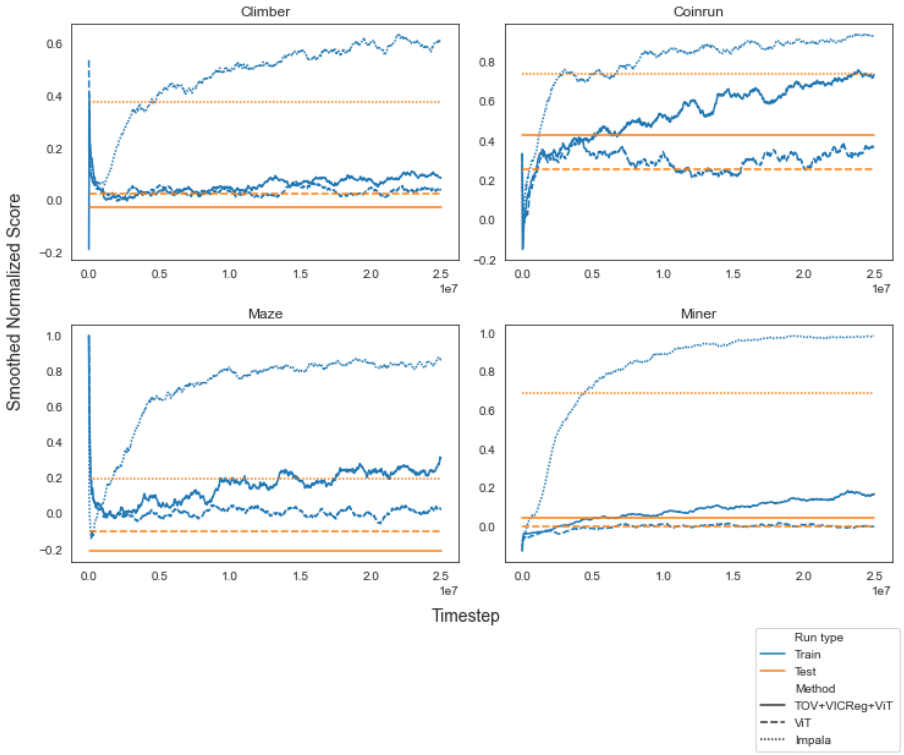


Figure 7.2: Average normalized learning curves and average normalized test scores for each game

by dirt blocks and gets more exposed every time the agent removes a dirt block. Climber, on the contrary, has a background that fills the majority of the frame and that moves downwards every time the agent climbs the level or jumps. So, such visual differences might be, in part, responsible for the result obtained, especially if we take into account how TOV-VICReg works. Our method during the pre-training phase tries to distil temporal information using three consecutive frames. While in the game Miner this will make the model attend to the agent and dirt blocks that the agent might remove in the game Climber the model will be prone to attend to elements in the background which are not relevant to solve the level.

Figure 7.4, shows attention maps across four timesteps at the game miner of ViT+TOV-VICReg. The

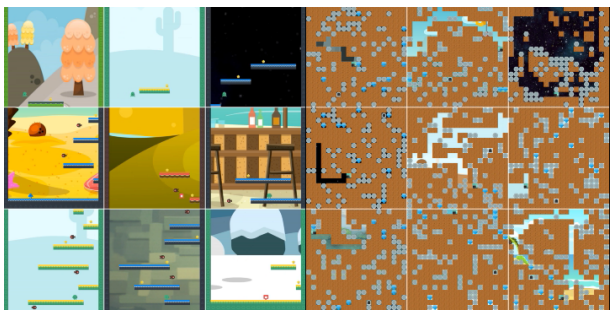


Figure 7.3: On the left we nine different levels from the game Climber and on the right nine levels from the game Miner

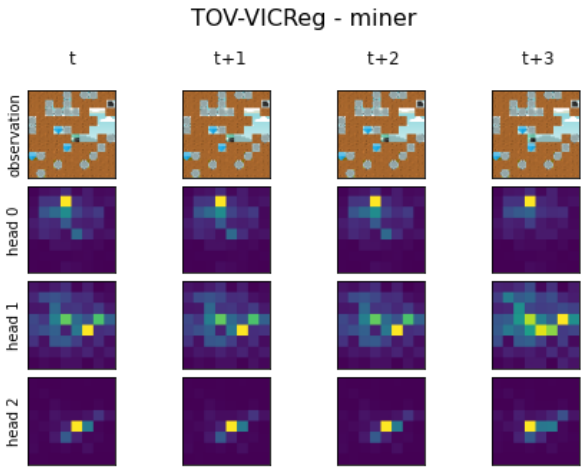


Figure 7.4

attention maps seem very localized and the different heads of the encoder appear to attend to different parts of the image, with head 1 attending to dirt blocks close to the agent including the block that the agent removes at t+3 and head 2 attending to the agent.

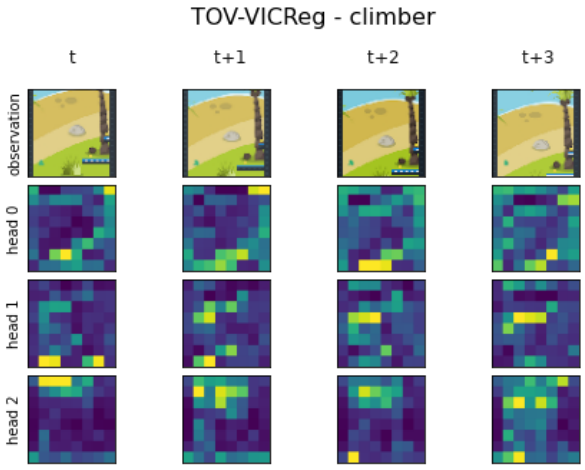


Figure 7.5

For the game Climber, Figure 7.5, the attention is much more spread across the image particularly the background, leaving the important features (platforms, player) unattended. While the agent during training can somewhat improve in the limited number of levels available when tested in unseen levels it most likely will fail due to poor representations given by the encoder that is not invariant to visual variations.

This analysis reveals a good hypothesis for the test score degradation and consequently a limitation of TOV-VICReg. Using such a small time window during the pretraining phase can negatively impact the

learned representations when non-important visual features, like random backgrounds, change during the level. We also hypothesise that an alternative to avoid such a problem could lie in using a larger time window like in SPR [66], where an RNN is trained to predict the representation of an observation k steps ahead, pushing the encoder to encode temporal information of the next k steps. This can result in an encoder being more invariant to elements that don't give the information needed to solve the level.

8

Conclusions

Contents

8.1 Reproducibility

Our work can be fully reproduced using the source code, pseudocode and hyperparameters listed below:

- Vision Transformer:
  - Source code: https://github.com/facebookresearch/dino/blob/main/vision_transformer.py
  - Hyperparameters: Appendix A.2
- ResNet: https://github.com/mila-iqia/SGI/blob/master/src/networks.py and Appendix A.3
- Self-supervised learning methods:
  - DINO: https://github.com/facebookresearch/dino
  - MoCo v3: https://github.com/facebookresearch/moco-v3
  - VICReg: https://github.com/facebookresearch/vicreg
  - TOV-VICReg: https://github.com/mgoulao/tov-vicreg
  - Hyperparameters for all methods: Appendix A.9
- Rainbow:
  - Source code: https://github.com/Kaixhin/Rainbow
  - Hyperparameters: Appendix A.10
- PPO:
  - Source code: https://github.com/vwxyzjn/cleanrl/blob/master/cleanrl/ppo_procgen.py
  - Hyperparameters: Appendix A.11
- RLiable: https://github.com/google-research/rliable
- Linear Probing for Reinforcement Learning evaluation task: https://github.com/mgoulao/Linear-Probing-for-RL

8.2 Future Work

Even though we applied the temporal order verification task to VICReg, it can be used with any method, including MoCo that shows good results even without this extension. Moreover, our generalization results showed limitations in TOV-VICReg that we believe are related to the small temporal window. Exploring self-supervised methods that consider larger temporal windows can lead to encoders more invariant to non-important information in the observations.

Regarding the proposed evaluation task, we believe that it can be useful for evaluating future representation learning methods. Our evaluation task showed limitations in the encoders produced by all self-supervised methods we studied, for example, being able to predict the action when the game has more than seven actions available. Overcoming these limitations might be an interesting research path which can culminate in more data-efficient Reinforcement Learning agents.

8.3 Discussion & Conclusions

In this work, we presented a study of ViT for vision-based deep reinforcement learning using self-supervised pretraining and proposed a self-supervised method that extends VICReg to better capture temporal relations between consecutive observations. Our results showed that the agent using a Vision Transformer that was pretrained with our method managed to surpass all other Vision Transformers, pretrained and non-pretrained, in sample efficiency and also achieves results very close to convolution-based models with far fewer parameters. Which reinforces the importance of encoding temporal relations between observations in the representation model, as shown by previous works, and also shows that even vision models with weaker inductive biases and more parameters, when pretrained with a competitive self-supervised method, can achieve similar results in sample efficiency.

Our results in generalization, unlike previous works [9], didn't show any improvement. Our understanding is that our dataset is not diverse enough to create a robust encoder that is more invariant and future work could explore using encoders that were first pretrained on more diverse datasets like ImageNet. In addition, we identified limitations in our method, which leads the encoder to pay attention to dynamic elements that are not important features for the gameplay, which is a result that should be considered by future self-supervised methods. Lastly, we only used procgen games which are designed for behaviour generalization using an infinite amount of levels with different sprites and layouts. Meaning that results in environments where the task remains the same and only the visual components change, like backgrounds, could be more positive.

Another important part of our work is the evaluations used and which can be a good reference for future work. We have presented three metrics to evaluate collapse and consequently, the quality of the representations learned with the different self-supervised learning methods during pretraining. Three different visualizations that give intuitions about the quality of the representations and some interpretability, which is the case of the attention maps. And a new linear probing evaluation task based on imitation learning, which can be very valuable for future work and even used as a benchmark when better validated.

Despite the focus of this work being the Vision Transformer, the results here shown should translate for convolution-based networks. Furthermore, the ability to use larger models, with millions of parame-

ters, that are as sample efficient (or more) as some of the most popular CNN-based models (like Nature CNN or Impala ResNet) and that can generalize to unseen observations is an important direction of research. Since it opens the door to using Deep RL in even more complex problems where smaller models (non-pretrained) tend to struggle to perform well. In this work, we try to advance the knowledge by studying the pretrain of a vision transformer using self-supervised methods. This approach has seen successes in natural language processing [46, 99], and computer vision [100]. Hence, we believe that similar approaches in RL have the potential to unlock new levels of performance [101].

Bibliography

[1] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015.

[2] N. Heess, D. TB, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. M. A. Eslami, M. Riedmiller, and D. Silver, “Emergence of Locomotion Behaviours in Rich Environments,” *arXiv:1707.02286*, Jul. 2017.

[3] OpenAI, I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, J. Schneider, N. Tezak, J. Tworek, P. Welinder, L. Weng, Q. Yuan, W. Zaremba, and L. Zhang, “Solving Rubik’s Cube with a Robot Hand,” *arXiv:1910.07113*, Oct. 2019.

[4] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, “Mastering the game of Go with deep neural networks and tree search,” *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016.

[5] M. G. Bellemare, S. Candido, P. S. Castro, J. Gong, M. C. Machado, S. Moitra, S. S. Ponda, and Z. Wang, “Autonomous navigation of stratospheric balloons using reinforcement learning,” *Nature*, vol. 588, no. 7836, pp. 77–82, Dec. 2020.

[6] J. Degrave, F. Felici, J. Buchli, M. Neunert, B. Tracey, F. Carpanese, T. Ewalds, R. Hafner, A. Abdolmaleki, D. de las Casas, C. Donner, L. Fritz, C. Galperti, A. Huber, J. Keeling, M. Tsimpoukelli, J. Kay, A. Merle, J.-M. Moret, S. Noury, F. Pesamosca, D. Pfau, O. Sauter, C. Sommariva, S. Coda, B. Duval, A. Fasoli, P. Kohli, K. Kavukcuoglu, D. Hassabis, and M. Riedmiller, “Magnetic control of tokamak plasmas through deep reinforcement learning,” *Nature*, vol. 602, no. 7897, pp. 414–419, Feb. 2022.

[7] G. Dulac-Arnold, D. Mankowitz, and T. Hester, “Challenges of Real-World Reinforcement Learning,” *arXiv:1904.12901*, Apr. 2019.

[8] M. Schwarzer, N. Rajkumar, M. Noukhovitch, A. Anand, L. Charlin, R. D. Hjelm, P. Bachman, and A. C. Courville, "Pretraining Representations for Data-Efficient Reinforcement Learning," in *Advances in Neural Information Processing Systems*, vol. 34, 2021, pp. 12 686–12 699.

[9] A. Stooke, K. Lee, P. Abbeel, and M. Laskin, "Decoupling Representation Learning from Reinforcement Learning," in *Proceedings of the 38th International Conference on Machine Learning*, Jul. 2021, pp. 9870–9879.

[10] D. Guo, B. A. Pires, B. Piot, J.-b. Grill, F. Altché, R. Munos, and M. G. Azar, "Bootstrap Latent-Predictive Representations for Multitask Reinforcement Learning," *arXiv:2004.14646*, Apr. 2020.

[11] D. Erhan, A. Courville, Y. Bengio, and P. Vincent, "Why Does Unsupervised Pre-training Help Deep Learning?" in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, Mar. 2010, pp. 201–208.

[12] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," in *International Conference on Learning Representations*, Sep. 2020.

[13] A. Bardes, J. Ponce, and Y. LeCun, "VICReg: Variance-Invariance-Covariance Regularization for Self-Supervised Learning," *arXiv:2105.04906*, May 2021.

[14] I. Misra, C. L. Zitnick, and M. Hebert, "Shuffle and Learn: Unsupervised Learning Using Temporal Order Verification," in *Computer Vision – ECCV 2016*, Cham, 2016, pp. 527–544.

[15] K. Cobbe, C. Hesse, J. Hilton, and J. Schulman, "Leveraging Procedural Generation to Benchmark Reinforcement Learning," *arXiv:1912.01588*, Jul. 2020.

[16] R. S. Sutton and A. G. Barto, *Reinforcement Learning, second edition: An Introduction*, Nov. 2018.

[17] OpenAi, "Spinning Up in Deep RL," 2019. [Online]. Available: https://spinningup.openai.com/en/latest/index.html

[18] S. Levine, A. Kumar, G. Tucker, and J. Fu, "Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems," *arXiv:2005.01643*, Nov. 2020.

[19] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. Lillicrap, and D. Silver, "Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model," *Nature*, vol. 588, no. 7839, pp. 604–609, Dec. 2020.

[20] W. Ye, S. Liu, T. Kurutach, P. Abbeel, and Y. Gao, "Mastering Atari Games with Limited Data," in *Advances in Neural Information Processing Systems*, vol. 34, 2021, pp. 25 476–25 488.

[21] Ö. Şimşek and A. G. Barto, “An intrinsic reward mechanism for efficient exploration,” in *Proceedings of the 23rd international conference on Machine learning*, ser. ICML ’06, New York, NY, USA, Jun. 2006, pp. 833–840.

[22] R. S. Sutton, D. Precup, and S. Singh, “Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning,” *Artificial Intelligence*, vol. 112, no. 1, pp. 181–211, Aug. 1999.

[23] P. Dayan and G. E. Hinton, “Feudal Reinforcement Learning,” in *Advances in Neural Information Processing Systems*, vol. 5, 1993.

[24] A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu, “FeUdal Networks for Hierarchical Reinforcement Learning,” *arXiv:1703.01161*, Mar. 2017.

[25] P.-L. Bacon, J. Harb, and D. Precup, “The Option-Critic Architecture,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, Feb. 2017.

[26] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, “Prioritized Experience Replay,” in *ICLR (Poster)*, Jan. 2016.

[27] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing Atari with Deep Reinforcement Learning,” *arXiv:1312.5602*, Dec. 2013.

[28] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, “Rainbow: Combining Improvements in Deep Reinforcement Learning,” *arXiv:1710.02298*, Oct. 2017.

[29] H. van Hasselt, A. Guez, and D. Silver, “Deep Reinforcement Learning with Double Q-Learning,” in *Thirtieth AAAI Conference on Artificial Intelligence*, Mar. 2016.

[30] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, “Dueling Network Architectures for Deep Reinforcement Learning,” in *Proceedings of The 33rd International Conference on Machine Learning*, Jun. 2016, pp. 1995–2003.

[31] M. G. Bellemare, W. Dabney, and R. Munos, “A Distributional Perspective on Reinforcement Learning,” Jan. 2017.

[32] M. Fortunato, M. G. Azar, B. Piot, J. Menick, M. Hessel, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin, C. Blundell, and S. Legg, “Noisy Networks For Exploration,” Feb. 2018.

[33] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal Policy Optimization Algorithms,” *arXiv:1707.06347*, Aug. 2017.

[34] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, “High-Dimensional Continuous Control Using Generalized Advantage Estimation,” *arXiv:1506.02438*, Oct. 2018.

[35] L. Engstrom, A. Ilyas, S. Santurkar, D. Tsipras, F. Janoos, L. Rudolph, and A. Madry, “Implementation Matters in Deep RL: A Case Study on PPO and TRPO,” Apr. 2020.

[36] “attention – APA Dictionary of Psychology.” [Online]. Available: https://dictionary.apa.org/attention

[37] D. Bahdanau, K. Cho, and Y. Bengio, “Neural Machine Translation by Jointly Learning to Align and Translate,” *arXiv:1409.0473*, May 2016.

[38] M.-T. Luong, H. Pham, and C. D. Manning, “Effective Approaches to Attention-based Neural Machine Translation,” *arXiv:1508.04025*, Sep. 2015.

[39] J. Cheng, L. Dong, and M. Lapata, “Long Short-Term Memory-Networks for Machine Reading,” *arXiv:1601.06733*, Sep. 2016.

[40] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is All you Need,” in *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[41] U. Khandelwal, H. He, P. Qi, and D. Jurafsky, “Sharp Nearby, Fuzzy Far Away: How Neural Language Models Use Context,” in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Melbourne, Australia, Jul. 2018, pp. 284–294.

[42] S. d’Ascoli, H. Touvron, M. Leavitt, A. Morcos, G. Biroli, and L. Sagun, “ConViT: Improving Vision Transformers with Soft Convolutional Inductive Biases,” *arXiv:2103.10697*, Jun. 2021.

[43] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov, “Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context,” *arXiv:1901.02860*, Jun. 2019.

[44] A. Fan, T. Lavril, E. Grave, A. Joulin, and S. Sukhbaatar, “Addressing Some Limitations of Transformers with Feedback Memory,” *arXiv:2002.09402*, Jan. 2021.

[45] Y. LeCun and I. Misra, “Self-supervised learning: The dark matter of intelligence.”

[46] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” *arXiv:1810.04805*, May 2019.

[47] J. Donahue, P. Krähenbühl, and T. Darrell, “Adversarial Feature Learning,” *arXiv:1605.09782*, Apr. 2017.

[48] C. Doersch, A. Gupta, and A. A. Efros, “Unsupervised Visual Representation Learning by Context Prediction,” *arXiv:1505.05192*, Jan. 2016.

[49] M. Noroozi and P. Favaro, “Unsupervised Learning of Visual Representations by Solving Jigsaw Puzzles,” *arXiv:1603.09246*, Aug. 2017.

[50] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, “A Simple Framework for Contrastive Learning of Visual Representations,” in *Proceedings of the 37th International Conference on Machine Learning*, Nov. 2020, pp. 1597–1607.

[51] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, “Momentum Contrast for Unsupervised Visual Representation Learning,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 9729–9738.

[52] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, “Extracting and composing robust features with denoising autoencoders,” in *Proceedings of the 25th international conference on Machine learning*, ser. ICML ’08, New York, NY, USA, Jul. 2008, pp. 1096–1103.

[53] S. Chopra, R. Hadsell, and Y. LeCun, “Learning a similarity metric discriminatively, with application to face verification,” in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05)*, vol. 1, Jun. 2005, pp. 539–546 vol. 1.

[54] R. Hadsell, S. Chopra, and Y. LeCun, “Dimensionality Reduction by Learning an Invariant Mapping,” in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’06)*, vol. 2, Jun. 2006, pp. 1735–1742.

[55] A. v. d. Oord, Y. Li, and O. Vinyals, “Representation Learning with Contrastive Predictive Coding,” *arXiv:1807.03748*, Jan. 2019.

[56] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” *arXiv:1512.03385*, Dec. 2015.

[57] X. Chen, H. Fan, R. Girshick, and K. He, “Improved Baselines with Momentum Contrastive Learning,” *arXiv:2003.04297*, Mar. 2020.

[58] M. Caron, H. Touvron, I. Misra, H. Jégou, J. Mairal, P. Bojanowski, and A. Joulin, “Emerging Properties in Self-Supervised Vision Transformers,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 9650–9660.

[59] G. Hinton, O. Vinyals, and J. Dean, “Distilling the Knowledge in a Neural Network,” *arXiv:1503.02531*, Mar. 2015.

[60] A. Zhan, P. Zhao, L. Pinto, P. Abbeel, and M. Laskin, "A Framework for Efficient Robotic Manipulation," *arXiv:2012.07975*, Dec. 2020.

[61] A. Anand, E. Racah, S. Ozair, Y. Bengio, M.-A. Côté, and R. D. Hjelm, "Unsupervised State Representation Learning in Atari," arXiv, Tech. Rep. arXiv:1906.08226, Nov. 2020.

[62] H. Wang, E. Miahi, M. White, M. C. Machado, Z. Abbas, R. Kumaraswamy, V. Liu, and A. White, "Investigating the Properties of Neural Network Representations in Reinforcement Learning," *arXiv:2203.15955*, Mar. 2022.

[63] Z. Yuan, Z. Xue, B. Yuan, X. Wang, Y. Wu, Y. Gao, and H. Xu, "Pre-Trained Image Encoder for Generalizable Visual Reinforcement Learning," in *First Workshop on Pre-training: Perspectives, Pitfalls, and Paths Forward at ICML 2022*, Jul. 2022.

[64] A. Srinivas, M. Laskin, and P. Abbeel, "CURL: Contrastive Unsupervised Representations for Reinforcement Learning," *arXiv:2004.04136*, Sep. 2020.

[65] X. Li, J. Shang, S. Das, and M. S. Ryoo, "Does Self-supervised Learning Really Improve Reinforcement Learning from Pixels?" *arXiv:2206.05266*, Jun. 2022.

[66] M. Schwarzer, A. Anand, R. Goel, R. D. Hjelm, A. Courville, and P. Bachman, "Data-Efficient Reinforcement Learning with Self-Predictive Representations," *arXiv:2007.05929*, May 2021.

[67] R. Agarwal, M. C. Machado, P. S. Castro, and M. G. Bellemare, "Contrastive Behavioral Similarity Embeddings for Generalization in Reinforcement Learning," *arXiv:2101.05265*, Mar. 2021.

[68] D. Yarats, R. Fergus, A. Lazaric, and L. Pinto, "Reinforcement Learning with Prototypical Representations," in *Proceedings of the 38th International Conference on Machine Learning*, Jul. 2021, pp. 11 920–11 931.

[69] M. Caron, I. Misra, J. Mairal, P. Goyal, P. Bojanowski, and A. Joulin, "Unsupervised Learning of Visual Features by Contrasting Cluster Assignments," in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 9912–9924.

[70] D. Yarats, I. Kostrikov, and R. Fergus, "Image Augmentation Is All You Need: Regularizing Deep Reinforcement Learning from Pixels," Feb. 2022.

[71] M. Laskin, K. Lee, A. Stooke, L. Pinto, P. Abbeel, and A. Srinivas, "Reinforcement Learning with Augmented Data," in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 19 884–19 895.

[72] T. Tao, D. Reda, and M. van de Panne, "Evaluating Vision Transformer Methods for Deep Reinforcement Learning from Pixels," *arXiv:2204.04905*, May 2022.

[73] N. Parthasarathy, S. M. A. Eslami, J. Carreira, and O. J. Hénaff, "Self-supervised video pretraining yields strong image representations," *arXiv:2210.06433*, Oct. 2022.

[74] T. D. Kulkarni, A. Gupta, C. Ionescu, S. Borgeaud, M. Reynolds, A. Zisserman, and V. Mnih, "Unsupervised Learning of Object Keypoints for Perception and Control," in *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[75] X. Wang and A. Gupta, "Unsupervised Learning of Visual Representations Using Videos," 2015, pp. 2794–2802.

[76] D. Pathak, R. Girshick, P. Dollar, T. Darrell, and B. Hariharan, "Learning Features by Watching Objects Move," 2017, pp. 2701–2710.

[77] Y. Yao, C. Liu, D. Luo, Y. Zhou, and Q. Ye, "Video playback rate perception for self-supervised spatio-temporal representation learning," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 6548–6557.

[78] H.-Y. Lee, J.-B. Huang, M. Singh, and M.-H. Yang, "Unsupervised representation learning by sorting sequences," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 667–676.

[79] D. Xu, J. Xiao, Z. Zhao, J. Shao, D. Xie, and Y. Zhuang, "Self-supervised spatiotemporal learning via video clip order prediction," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 10 334–10 343.

[80] U. Ahsan, R. Madhok, and I. Essa, "Video jigsaw: Unsupervised learning of spatiotemporal context for video action recognition," in *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2019, pp. 179–189.

[81] C. Sun, A. Myers, C. Vondrick, K. Murphy, and C. Schmid, "Videobert: A joint model for video and language representation learning," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 7464–7473.

[82] C. Feichtenhofer, H. Fan, B. Xiong, R. Girshick, and K. He, "A large-scale study on unsupervised spatiotemporal representation learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 3299–3309.

[83] G. Lorre, J. Rabarisoa, A. Orcesi, S. Ainouz, and S. Canu, "Temporal contrastive pretraining for video action recognition," in *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, 2020, pp. 662–670.

[84] J. Knights, B. Harwood, D. Ward, A. Vanderkop, O. Mackenzie-Ross, and P. Moghadam, "Temporally coherent embeddings for self-supervised video representation learning," in *2020 25th International Conference on Pattern Recognition (ICPR)*. IEEE, 2021, pp. 8914–8921.

[85] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. H. Richemond, E. Buchatskaya, C. Doersch, B. A. Pires, Z. D. Guo, M. G. Azar, B. Piot, K. Kavukcuoglu, R. Munos, and M. Valko, "Bootstrap your own latent: A new approach to self-supervised Learning," *arXiv:2006.07733*, Sep. 2020.

[86] X. Chen, S. Xie, and K. He, "An Empirical Study of Training Self-Supervised Vision Transformers," *arXiv:2104.02057*, Aug. 2021.

[87] R. Wightman, "Pytorch image models," https://github.com/rwightman/pytorch-image-models, 2019.

[88] R. Agarwal, D. Schuurmans, and M. Norouzi, "An Optimistic Perspective on Offline Reinforcement Learning," *arXiv:1907.04543*, Jun. 2020.

[89] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[90] Ł. Kaiser, M. Babaeizadeh, P. Miłos, B. Osiński, R. H. Campbell, K. Czechowski, D. Erhan, C. Finn, P. Kozakowski, S. Levine, A. Mohiuddin, R. Sepassi, G. Tucker, and H. Michalewski, "Model Based Reinforcement Learning for Atari," Sep. 2019.

[91] R. Agarwal, M. Schwarzer, P. S. Castro, A. C. Courville, and M. Bellemare, "Deep Reinforcement Learning at the Edge of the Statistical Precipice," in *Advances in Neural Information Processing Systems*, 2021.

[92] H. P. van Hasselt, M. Hessel, and J. Aslanides, "When to use parametric models in reinforcement learning?" in *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[93] K. P. Kielak, "Do recent advancements in model-based deep reinforcement learning really improve data efficiency?" 2019.

[94] K. Arulkumaran, "Rainbow," Aug. 2022.

[95] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," *arXiv:1606.01540*, Jun. 2016.

[96] L. v. d. Maaten and G. Hinton, "Visualizing Data using t-SNE," *Journal of Machine Learning Research*, vol. 9, no. 86, pp. 2579–2605, 2008.

[97] S. Huang, R. F. J. Dossa, C. Ye, and J. Braga, “CleanRL: High-quality Single-file Implementations of Deep Reinforcement Learning Algorithms,” *arXiv:2111.08819*, Nov. 2021.

[98] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, S. Legg, and K. Kavukcuoglu, “IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures,” *arXiv:1802.01561*, Jun. 2018.

[99] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, “Language Models are Few-Shot Learners,” in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 1877–1901.

[100] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever, “Learning Transferable Visual Models From Natural Language Supervision,” in *Proceedings of the 38th International Conference on Machine Learning*, Jul. 2021, pp. 8748–8763.

[101] B. Baker, I. Akkaya, P. Zhokhov, J. Huizinga, J. Tang, A. Ecoffet, B. Houghton, R. Sampedro, and J. Clune, “Video PreTraining (VPT): Learning to Act by Watching Unlabeled Online Videos,” *arXiv:2206.11795*, Jun. 2022.

A

Appendix

A.1 ViT's patch size

The patch size of a Vision Transformer can largely affect the performance of the model and the number of computations per data sample. A patch size of 1 is equivalent to using the pixels as tokens while a patch size of 16 converts patches of 16x16 pixels to a single token, i.e. the hyperparameter affects the number of tokens quadratically. On the other hand, larger patches might not allow the model to learn as good representations, therefore it is necessary to find a patch size that balances the computational cost with task performance. For this work, we explored several different sizes and evaluated their data efficiency by training Rainbow in the Atari game MsPacman for 100k across 10 different seeds. Our results, Table A.1, show marginal differences between a patch size of 8 and 10, however, we didn't observe a significant difference in the training time and for that reason we decided to use a patch size of 8 for all our experiments.

Patch size	Score	Mean Time
6	305.7 ± 71.0	4:56.33
8	801.9 ± 523.9	3:22.4
10	778.0 ± 324.0	3:10.2
12	627.0 ± 284.0	3:12.8

Table A.1: Scores obtained for different patch sizes

A.2 Vision Transformer hyperparameters

In this work, we used the ViT tiny which corresponds to using the hyperparameters at Table A.2.

Hyperparameter	Value
Patch Size	8
Embedding dimension	192
Depth	12
Number of heads	3
MLP ratio	4
Use bias in QKV	True
Normalization	Layer Normalization
Normalization: epsilon	1.0×10^{-6}
Use Dropout	False

Table A.2: Hyperparameters used for the Vision Transformer

A.3 ResNet architecture

The ResNet we used is based on the ResNet used for SGI, which uses three inverted residual blocks with an expansion ratio of two, where each block is a sequence of Conv2D, Batch Normalization, and ReLU, as shown in Figure A.1. However, to have a number of parameters similar to the ViT tiny we added an additional residual block and changed the channels of each block to 64, 128, 256 and 512. Additionally, we change the strides of each block to 2 for all blocks. The encoder computes representations vectors with size of 18432.

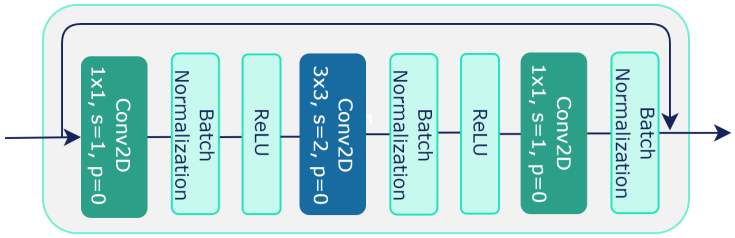


Figure A.1: ResNet residual block

A.4 Models sizes

Model Name	# parameters
Nature CNN	75.936
ResNet	4.932.524
ViT tiny	5.526.720

Table A.3: Number of learnable parameters of each model we used

A.5 Gym's wrappers setup python code

```
1    env = AtariPreprocessing(env, terminal_on_life_loss=True, scale_obs=True)
2    env = TransformReward(env, np.sign)
3    env = FrameStack(env, 3)
```

Listing A.1: Gym Atari Wrappers

A.6 PPO derivations

$$
\begin{aligned}
\nabla_\theta J(\pi_\theta)|_{\theta_t} &= \nabla_\theta \mathop{\mathbb{E}}_{\tau\sim\rho_\theta}[R(\tau)] \\
&= \nabla_\theta \int_\tau \rho_\theta(\tau) R(\tau) \\
&= \int_\tau \nabla_\theta \rho_\theta(\tau) R(\tau) \\
&= \int_\tau \rho_\theta(\tau) \frac{\nabla_\theta \rho_\theta(\tau)}{\rho_\theta(\tau)} R(\tau) \\
&= \int_\tau \rho_\theta(\tau) \nabla_\theta \log \rho_\theta(\tau) R(\tau) \\
&= \mathop{\mathbb{E}}_{\tau\sim\rho_\theta}[\nabla_\theta \log \rho_\theta(\tau) R(\tau)] \\
&= \mathop{\mathbb{E}}_{\tau\sim\rho_\theta}\left[\nabla_\theta \log \left(P(s_0) \prod_{t=0}^{T} \pi_\theta(a_t|s_t) P(s_{t+1}|s_t, a_t) \right) R(\tau)\right] \\
&= \mathop{\mathbb{E}}_{\tau\sim\rho_\theta}\left[\nabla_\theta \log P(s_0) + \sum_{t=0}^{T} \left(\log \pi_\theta(a_t|s_t) + \log P(s_{t+1}|s_t, a_t)\right) R(\tau)\right] \\
&= \mathop{\mathbb{E}}_{\tau\sim\rho_\theta}\left[\nabla_\theta \sum_{t=0}^{T} \left(\log \pi_\theta(a_t|s_t)\right) R(\tau)\right] \\
&= \mathop{\mathbb{E}}_{\tau\sim\rho_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \left(\pi_\theta(a_t|s_t)\right) R(\tau)\right]
\end{aligned}
\tag{A.1}
$$

$$
\begin{aligned}
J(\pi_{\theta'}) - J(\pi_\theta) &= J(\pi_{\theta'}) - \mathbb{E}_{s_0 \sim \rho_0}\left[V^{\pi_\theta}(s_0)\right] \\
&= J(\pi_{\theta'}) - \mathbb{E}_{\tau \sim \rho_{\theta'}}\left[V^{\pi_\theta}(s_0)\right] \\
&= J(\pi_{\theta'}) - \mathbb{E}_{\tau \sim \rho_{\theta'}}\left[\sum_{t=0}^{\infty} \gamma^t V^{\pi_\theta}(s_t) - \sum_{t=1}^{\infty} \gamma^t V^{\pi_\theta}(s_t)\right] \\
&= J(\pi_{\theta'}) + \mathbb{E}_{\tau \sim \rho_{\theta'}}\left[\sum_{t=0}^{\infty} \gamma^t \left(\gamma V^{\pi_\theta}(s_{t+1}) - V^{\pi_\theta}(s_t)\right)\right] \\
&= \mathbb{E}_{\tau \sim \rho_{\theta'}}\left[R(\tau)\right] + \mathbb{E}_{\tau \sim \rho_{\theta'}}\left[\sum_{t=0}^{\infty} \gamma^t \left(\gamma V^{\pi_\theta}(s_{t+1}) - V^{\pi_\theta}(s_t)\right)\right] \\
&= \mathbb{E}_{\tau \sim \rho_{\theta'}}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right] + \mathbb{E}_{\tau \sim \rho_{\theta'}}\left[\sum_{t=0}^{\infty} \gamma^t \left(\gamma V^{\pi_\theta}(s_{t+1}) - V^{\pi_\theta}(s_t)\right)\right] \\
&= \mathbb{E}_{\tau \sim \rho_{\theta'}}\left[\sum_{t=0}^{\infty} \gamma^t \left(r(s_t, a_t) + \gamma V^{\pi_\theta}(s_{t+1}) - V^{\pi_\theta}(s_t)\right)\right] \\
&= \mathbb{E}_{\tau \sim \rho_{\theta'}}\left[\sum_{t=0}^{\infty} \gamma^t \left(Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t)\right)\right] \\
&= \mathbb{E}_{\tau \sim \rho_{\theta'}}\left[\sum_{t=0}^{\infty} \gamma^t A^{\pi_\theta}(s_t, a_t)\right]
\end{aligned}
\tag{A.2}
$$

A.7 TOV-VICReg Pseudocode

```
# N: batch size, D: dimension of the embedding
# mse_loss: Mean square error loss function, off_diagonal: off-diagonal
    elements of a matrix, relu: ReLU activation function
# shuffle: shuffles elements in a certain dimension according to a
    permutation index
for u, v, w in loader: # load a batch with N samples
    # u -> x_{t}
    # v -> x_{t-1}
    # w -> x_{t+1}

    # apply augmentations
    u_a  = augmentation_1(u)
    u_b  = augmentation_2(u)
    v  = augmentation_3(v)
    w  = augmentation_3(w)

```

```
    # compute representations
    y_u_a = encoder(u_a)
    y_u_b = encoder(u_b)
    y_v = encoder(v)
    y_w = encoder(w)

    # compute embeddings
    z_u_a = expander(y_u_a)
    z_u_b = expander(y_u_b)
    z_v = expander(y_v)
    z_w = expander(y_w)

    shuffle_indexes = randint(0, 6) # sample from 0 to 3 permutations of 3
    labels = where(shuffle_indexes == 0, 0, 1)

    # concat and shuffle (N, 3, D)
    c = concat(p_u_a, p_v, p_w)
    c = shuffle(c, shuffle_indexes, dim=1)

    # temporal loss
    preds = linear(c) # Linear layer Dx6
    temp_loss = Binary_Cross_Entropy_Loss(preds, labels)

    # invariance loss
    sim_loss = mse_loss(z_a, z_b)

    # variance loss
    std_z_a = torch.sqrt(z_a.var(dim=0) + 1e-04)
    std_z_b = torch.sqrt(z_b.var(dim=0) + 1e-04)
    std_loss = torch.mean(relu(1 - std_z_a)) + torch.mean(relu(1 - std_z_b))

    # covariance loss
    z_a = z_a - z_a.mean(dim=0)
    z_b = z_b - z_b.mean(dim=0)
    cov_z_a = (z_a.T @ z_a) / (N - 1)
    cov_z_b = (z_b.T @ z_b) / (N - 1)
    cov_loss = off_diagonal(cov_z_a).pow_(2).sum() / D + \
               off_diagonal(cov_z_b).pow_(2).sum() / D
```

```
53
54      # loss
55      loss = inv_coef * inv_loss \
56           + var_coef * var_loss \
57           + cov_coef * cov_loss \
58           + temp_coef * temp_loss
59
60      # optimization step
61      loss.backward()
62      optimizer.step()
```

Listing A.2: Pytorch-like TOV-VICReg pseudocode

A.8 TOV-VICReg augmentations

```
1  # Augmentation 1 / tau
2  RandomResizedCrop(84, scale=(0.08, 1.)),
3  RandomApply([
4      ColorJitter(0.4, 0.4, 0.2, 0.1)
5  ], p=0.8),
6  RandomGrayscale(p=0.2),
7  RandomApply([GaussianBlur((7, 7), sigma=(.1, .2))], p=1.0),
8  RandomHorizontalFlip()
9
10 # Augmentation 2 / tau prime
11 RandomResizedCrop(84, scale=(0.08, 1.)),
12 RandomApply([
13     ColorJitter(0.4, 0.4, 0.2, 0.1)
14 ], p=0.8),
15 RandomGrayscale(p=0.2),
16 RandomApply([GaussianBlur((7, 7), sigma=(.1, .2))], p=0.1),
17 RandomSolarize(120, p=0.2),
18 RandomHorizontalFlip(),
19
20 # Augmentation 3 / tau two prime and tau three prime
21 RandomApply([
```

```
    ColorJitter(0.4, 0.4, 0.2, 0.1)
], p=0.8),
RandomGrayscale(p=0.2),
```

Listing A.3: Pytorch-like pseudocode of TOV-VICReg augmentations

A.9 Self-Supervised methods hyperparameters

Hyperparameter	Value
Drop path rate	0.1
Freeze last layer	True
# local crops	8
Local crops scale interval	[0.05, 0.5]
Learning rate	5.0×10^{-4}
Min learning rate	1.0×10^{-6}
Teacher ema coefficient	0.996
Normalize last layer	False
Optimizer	AdamW
Out dimension	1024
Use batch normalization in head	false
Teacher warmup temperature	0.04
# warmup epochs for teacher temperature	0
Weight decay	0.04
Weight decay final value	0.4

Table A.4: DINO hyperparameters

Hyperparameter	Value
Random crop min scale	0.08
Learning rate	0.6
Number of features	256
Momentum encoder ema coefficient	0.99
MLP hidden dimensions	4096
Softmax temperature	1.0
Optimizer	LARS
Weight decay	1.0×10^{-6}

Table A.5: MoCo v3 hyperparameters

Hyperparameter	Value
Base Learning Rate	0.2
Weight decay	1.0×10^{-6}
MLP dimensions	1024-1024-1024
Invariance coefficient	25.0
Variance coefficient	25.0
Covariance coefficient	1.0

Table A.6: VICReg hyperparameters

Hyperparameter	Value
Base Learning Rate	0.2
Weight decay	1.0×10^{-6}
MLP dimensions	1024-1024-1024
Invariance coefficient	25.0
Variance coefficient	25.0
Covariance coefficient	10.0

Table A.7: TOV-VICReg hyperparameters

A.10 Data-Efficient Rainbow hyperparameters

Hyperparameter	value
Grey-scaling	True
Observation down-sampling	(84, 84)
Frames stacked	4
Action repetitions	4
Reward clipping	[-1, 1]
Terminal on loss of life	True
Max frames per episode	108K
Update	Distributional Double Q
Target network update period	every 2000 updates
Support of Q-distribution	51 bins
Discount factor	0.99
Minibatch size	32
Optimizer	Adam
Optimizer: first moment decay	0.9
Optimizer: second moment decay	0.999
Optimizer:	0.00015
Max gradient norm	10
Priority exponent	0.5
Priority correction	$0.4 \rightarrow 1$
Noisy nets parameter	0.1
Min replay size for sampling	1600
Memory size	unlimited
Replay period every	1 step
Multi-step return length	20
Q network: hidden units	512
Optimizer: noisy nets learning rate	0.0001
Optimizer: encoder learning rate	0.000001

Table A.8: Hyperparameters used for Rainbow

A.11 PPO hyperparameters

Hyperparameter	value
# timesteps	25 000 000
# parallel environments	64
# step per rollout	256
Optimizer	Adam
Optimizer: moment decay	0.00005
Optimizer: linear layers learning rate	0.0005
Optimizer: encoder learning rate	0.000005
Value loss coefficient	0.5
Entropy loss coefficient	0.01
Clip Value Loss	True
Gradient clipping max norm	0.5
Normalized advantage	True
Policy update # epochs	3
# mini batches	8
GAE lambda	0.95
Discount factor	0.999
Anneal learning rate	False

Table A.9: Hyperparameters used for PPO

A.12 SSL methods computational performance

Table A.10 shows a comparison of the total time that took to train a ViT tiny using the different self-supervised learning methods. We also show the number of workers used to load the batches from memory, which includes applying the augmentations, and the batch size. Both hyperparameters impact the total time and ideally, we would have liked to use the same values for all methods. However, in the case of the batch size, this value needs to be adjusted to keep the VRAM used by the training script below 30GB.

Method	# workers	batch size	Time
DINO	10	160	2d 18h 31m 55s
MoCov3	8	1024	15h 28m 42s
VICReg	8	1024	15h 48m 45s
TOV-VICReg	8	512	18h 53m 32s

Table A.10: SSL methods computational performance comparison

A.13 CleanRL PPO benchmark

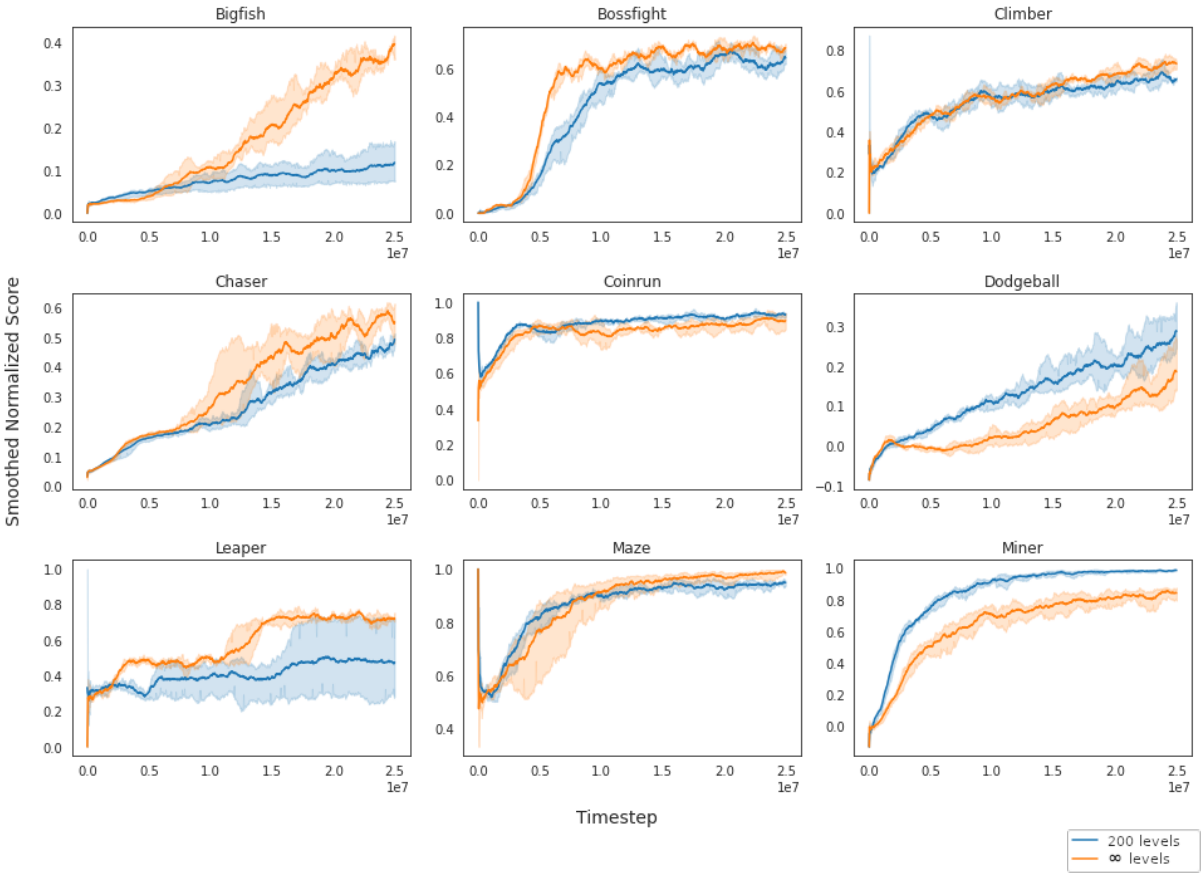


Figure A.2

A.14 RL data-efficiency results table

Games	Nature CNN	ResNet	ViT	ViT+TOV-VICReg	ViT+DINO	ViT+MoCo	ViT+VICReg
Assault	355.1 ± 105.2	452.0 ± 349.1	322.7 ± 146.9	366.3 ± 124.5	**493.3 ± 254.7**	493.3 ± 181.1	408.5 ± 156.6
Alien	210.8 ± 133.1	186.9 ± 104.4	250.6 ± 142.6	197.6 ± 114.4	275.1 ± 153.2	**380.8 ± 194.7**	187.3 ± 118.8
Bank Heist	37.6 ± 29.5	30.6 ± 18.6	**58.3 ± 115.4**	34.5 ± 18.8	18.6 ± 10.7	21.0 ± 30.1	29.6 ± 13.6
Breakout	**5.1 ± 3.3**	4.7 ± 2.1	3.2 ± 2.6	4.3 ± 2.7	2.8 ± 2.1	2.7 ± 1.6	3.1 ± 1.6
Chopper Command	828.0 ± 323.8	737.0 ± 354.0	747.0 ± 268.5	853.0 ± 312.2	760.0 ± 249.0	**968.0 ± 673.0**	668.0 ± 274.9
Freeway	**30.4 ± 1.2**	26.5 ± 2.5	21.2 ± 1.4	25.9 ± 2.7	25.0 ± 2.0	22.5 ± 2.1	23.7 ± 2.4
Frostbite	120.1 ± 25.9	107.9 ± 26.8	127.5 ± 15.6	**143.7 ± 106.7**	132.7 ± 14.1	111.3 ± 37.0	120.0 ± 18.2
Kangaroo	**776.0 ± 1035.4**	405.0 ± 226.4	60.0 ± 91.7	704.0 ± 1076.7	316.0 ± 233.5	384.0 ± 531.0	268.0 ± 244.5
MsPacman	**781.3 ± 417.1**	757.7 ± 413.2	618.9 ± 259.9	639.5 ± 378.4	698.9 ± 374.5	586.4 ± 257.5	633.0 ± 372.1
Pong	-13.6 ± 9.7	-12.0 ± 8.6	-21.0 ± 0.0	**-6.2 ± 13.4**	-18.4 ± 3.4	-17.6 ± 4.3	-15.1 ± 3.9

Table A.11: Table of results (mean and standard error) from experiments presented in Chapter 5.The bold values represent the best scores for the corresponding game