Pretraining the Vision Transformer using self-supervised methods for vision based Deep Reinforcement Learning

Manuel Goulão Instituto Superior Técnico / INESC-ID
manuel.silva.goulao@tecnico.ulisboa.pt

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. Our results show that the vision transformer, when pretrained with TOV-VICReg, outperforms the other self-supervised methods but still struggles to overcome a CNN. Nevertheless, we were able to outperform a CNN in two of the ten games where we perform a 100k steps evaluation. 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. [1]

1 Introduction

Despite the successes of deep reinforcement learning agents in the last decade, these still require a large amount of data or interactions to learn good policies. This data inefficiency makes current methods difficult to apply to environments where interactions are more expensive or data is scarce, which is the case in many real-world applications. 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 (Schwarzer et al., 2021b; Stooke et al., 2021; Guo et al., 2020), 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. A common

[1] Work done under the supervision of Arlindo Oliveira

Deep Reinforcement Learning Workshop, 36th Conference on Neural Information Processing Systems (NeurIPS 2022).

approach to evaluating such methods is to train a network composed of the pretrained encoder, with the parameters frozen, paired with a linear layer in popular datasets, like ImageNet. These evaluations have shown that these methods can achieve high scores in different benchmarks, which shows how well the current state-of-the-art methods are able to 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) adds robustness to the network and gives better generalization capabilities (Erhan et al., 2010).

Also recently, a new architecture for vision-based tasks called the Vision Transformer (ViT) (Dosovitskiy et al., 2020) has shown impressive results in several benchmarks without using any convolutions. This architecture presents much weaker inductive biases when compared to a CNN, which can result in lower data efficiency. But 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. Nonetheless, and 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 state-of-the-art (SOTA) self-supervised learning methods. Consequently, we propose TOV-VICReg (Temporal Order Verification-VICReg) which is an extension of VICReg (Variance Invariance Covariance Regularization) (Bardes et al., 2022) that adds a temporal order verification task (Misra et al., 2016) to help the model better capture the temporal relations between consecutive observations. 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 7. After our empirical results in the Atari games, we present a small study of the pretrained encoders using several metrics to understand if they suffer from any representational collapse and also analyse the learned representations using similarity matrices and attention maps.

Our main contributions are:

- We propose a new self-supervised learning method which extends VICReg to capture the temporal relations between consecutive frames through a temporal order verification task, in Section 4.
- We pretrain a Vision Transformer using several SOTA self-supervised methods and our proposed method, and study them through metrics (Section 7), visualizations (Section 8) and fine-tuning in reinforcement learning (Section 6), where we show that temporal relations learned by the model pretrained with our method contribute to a great increase in data efficiency.

2 Related Work

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 fine-tuning phase (Schwarzer et al., 2021b; Zhan et al., 2020) or superior results in evaluation tasks, like AtariARI (Anand et al., 2020). Others have pretrained representations using RL algorithms, like DQN, and transfer those learned representations to a new learning task (Wang et al., 2022).

Temporal Relations Other works have explored learning representations that have temporal information encoded. ATC (Augmented Temporal Contrast) (Stooke et al., 2021) trains an encoder to compute temporally consistent representations using contrastive learning, and the ST-DIM (SpatioTemporal DeepInfoMax) (Anand et al., 2020) captures spatial-temporal information by maximizing the mutual information between features of two consecutive observations.

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 (Srinivas et al., 2020) adds a contrastive loss using a siamese network with a momentum encoder. Another work studies different joint-learning frameworks using different self-supervised methods (Li et al., 2022). SPR (Schwarzer et al., 2021a) 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 (Agarwal et al., 2021a) 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 (Guo et al., 2020) 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 (Yarats et al., 2021) uses an auxiliary self-supervised objective to learn representations and prototypes (Caron et al., 2020), and uses the learned prototypes to compute intrinsic rewards which will push the agent to explore the environment.

Augmentations While we only use augmentations in the pre-training phase, their use during reinforcement learning has also been studied. Methods like DrQ (Kostrikov et al., 2021) and RAD (Laskin et al., 2020) pair an RL algorithm, like SAC, with image augmentations to improve data efficiency and generalization of the algorithms.

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 (Tao et al., 2022).

3 Background

3.1 Vision Transformer

ViT (Dosovitskiy et al., 2020) is a model, for image classification tasks, that doesn't rely on CNNs using only attention. The model wraps the encoder of a Transformer, uses patches of the input image as tokens and adds a classification token which after the computation will serve as the image representation. When compared to CNNs, ViT presents weaker image-specific inductive biases which allow the CNNs for much sample-efficient learning (d'Ascoli et al., 2021), although it has been shown that with enough data the image-specific inductive biases become less important (Dosovitskiy et al., 2020).

3.2 Reinforcement Learning

The problem of an **agent** learning to solve a 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 $\mathcal{T}(s', s, a)$. 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, $\pi(a_t|s_t)$. 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.

3.2.1 DQN and Rainbow

DQN (Mnih et al., 2013) is a value-based method and uses a network with parameters ϕ that given a state s outputs a prediction of the distribution of Q values over actions, $Q_{\phi}(s, a)$. The network learns the Q function by minimizing the mean squared error: $(y - Q_{\phi}(s, a))^2$, where $y = r + \gamma \, max_{a'} Q_{\phi}(s', a')$. The algorithm has the following structure:

1. Start episode = 1 and repeat
   (a) Start t=1 and repeat T time:
      i. With probability ϵ: $a_t = random()$, otherwise: $a_t = argmax_{a'} \, Q(s, a')$
      ii. Execute a_t and observe s'_t and r_t
      iii. Store transition $\{s_t, a_t, r_t, s'_t\}$ in the replay buffer $\mathcal{D}$
      iv. Sample a mini-batch of transitions $\{s_j, a_j, r_j, s'_j\}$ from $\mathcal{D}$

v. $y_j = r_j + \gamma \, max_{a'_j} Q_\phi(s'_j, a'_j)$

vi. $\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi(s_j, a_j)}{d\phi}(Q_\phi(s_j, a_j) - y_j)$

Several works followed the DQN algorithm which introduced changes to improve performance. Rainbow (Hessel et al., 2017) combines six improvements, Double Q-Learning (van Hasselt et al., 2016), Prioritized Replay Schaul et al. (2016), Dueling Networks (Wang et al., 2016), Multi-step Learning (Sutton & Barto, 2018), Distributional RL (Bellemare et al., 2017), and Noisy Nets (Fortunato et al., 2018) resulting in a more stable and sample efficient algorithm.

3.3 Self-Supervised methods

Recent self-supervised methods for vision tasks can be put in two main categories: contrastive and non-contrastive.

In contrastive learning, methods like MoCo (He et al., 2020) or SimCLR (Chen et al., 2020a) learn using a loss function that pulls the positive samples together and pushes the negative samples apart. These methods usually require very large batch sizes or auxiliary structures that allow for more negative samples. MoCo, in particular, has three iterations v1 (He et al., 2020), v2 (Chen et al., 2020b), and v3(Chen et al., 2021). In this work, we consider the more recent version (v3). This version uses a siamese network, where in one path the augmented samples (queries) are computed by an encoder f_θ (backbone) and a projector g_ϕ, and in the other the samples (keys) by a momentum-encoder $f_\theta{}'$ and a projector g_ϕ. The loss function is the InfoNCE loss, with temperature, of the dot product of the queries with the keys.

On the other hand, non-contrastive methods don't rely on the notion of positive and negative samples which results in a vast number of different approaches. DINO (Caron et al., 2021) consists of a siamese network where each path is fed with a random augmentation of the input and where the encoders learn to minimize the cross-entropy between their normalized output probability distributions, computed using a softmax with temperature scaling. The teacher encoder is updated using an exponential moving average of the student encoder parameters and in its computation path is used an additional centring operation that contributes to an asymmetry that helps the method avoid collapse. 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.

VICReg, on the other hand, tries to learn representations invariant to augmentations by minimizing the L2 distance while maintaining some variance in the representation features and decorrelating features. A more detailed explanation of the method will be presented in Section 4.

For this study we selected DINO, MoCo, and VICReg since they are currently considered state-of-the-art, their official implementations are available in PyTorch, and each represents a different type of approach.

4 TOV-VICReg

VICReg is a non-contrastive method that trains a network to be invariant to augmentations applied to the inputs while avoiding a trivial solution with the help of two additional losses, called variance and covariance, 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.

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 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 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 3. 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.

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

$$v(Z) = \frac{1}{d} \sum_{j}^{d} \max(0, \gamma - \sqrt{Var(Z^j)}) \tag{2}$$

$$c(Z) = \frac{1}{d} \sum_{i \neq j} [\text{Cov(Z)}]_{i,j}^2 \tag{3}$$

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 (Misra et al., 2016), 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 and in most of our experiments, the value is 0.1. Figure 1 visually illustrates TOV-VICReg.

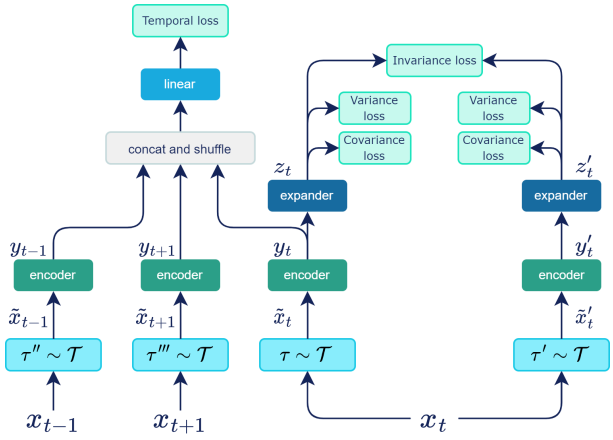

Figure 1: TOV-VICReg architecture

At each step we sample 3 consecutive observations, $\{x_{t-1},\ x_t,\ x_{t+1}\}$, the x_t is processed by two different augmentations, and like VICReg these are the augmentations used in BYOL (Grill et al., 2020), while x_{t-1} and x_{t+1} are processed by two simple augmentations composed of a color jitter and a random grayscale. 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 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.

5 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 (Chen et al., 2021), DINO (Caron et al., 2021) and VICReg (Bardes et al., 2022). 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). 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 (Agarwal et al., 2020). For each game, we use three checkpoints with a size of 100 thousand data points (observations), which makes up a total of 3 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 F contains the tables with the hyperparameters used for each method.

6 Data-Efficiency

To test the pretrained Vision Transformers in reinforcement learning and compare data-efficiency gains, we trained in the 10 games used for pre-training for 100k steps using the Rainbow algorithm (Hessel et al., 2017), with the DER (van Hasselt et al., 2019) hyperparameters. The only difference between the agents at the start is the representation module. We chose two networks to compare against, the Nature CNN (Mnih et al., 2015), and the SGI ResNet Large which is a larger version of the ResNet used in the SGI method (Schwarzer et al., 2021b) that has a size roughly similar to the ViT tiny. Moreover, we use a learning rate two orders of magnitude smaller for the encoder (1×10^{-6}), which previous works and experiments performed by us show to be beneficial (Schwarzer et al., 2021b).

In this section, to report our results we follow the rliable (Agarwal et al., 2021b) evaluation framework, where the scores of all games are normalized and treated as one single task.

6.1 Results

Figure 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 Table 9). 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.

7 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.

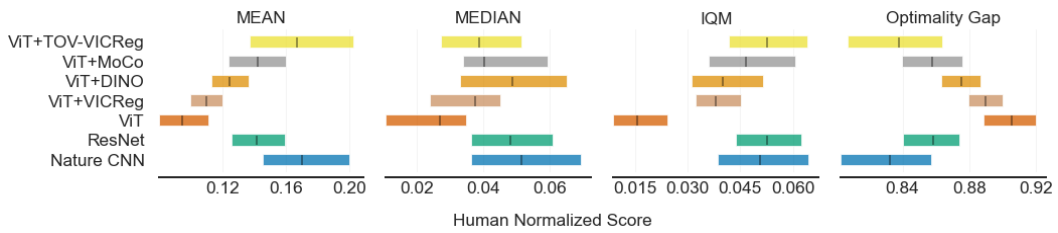

Figure 2: The eval runs across the different games are normalized and treated as a single task. The IQM corresponds to the Inter-Quartile Mean among all the runs, where the top and bottom 25% are discarded and the mean is calculated over the remaining 50%. The Optimality Gap refers to the number of runs that fail to surpass the human average score, i.e. 1.0.

Dimensional Collapse All methods seem to avoid dimensional collapse, i.e. most dimensions have a singular value larger than zero, as observed in Figure 3. 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.

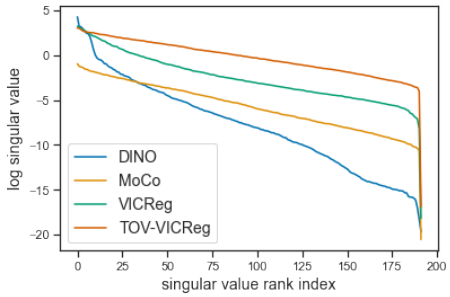

Figure 3: Logarithm of the singular values of the representation vector's covariance matrix sorted by value.

Representational Collapse Results in Table 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 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 1: Average standard deviation of the representation vector

Informational Collapse We report in Table 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 2: Average correlation coefficient

8 Representations

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.

Cosine similarity Figure 4 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, Figure 5. 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 7. 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 5. Given the empirical results, we believe that this capacity to distinguish observations from the same game might be a good indicator.

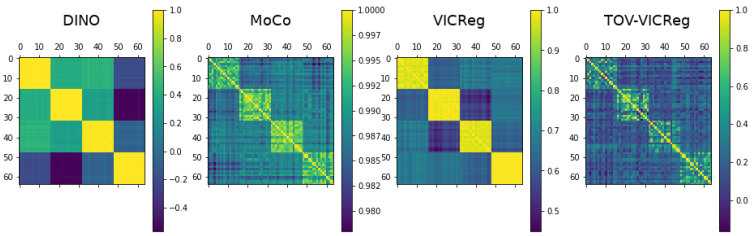

Figure 4: Similarity matrices of the representations computed by MoCo, DINO, VICReg, and TOV-VICReg respectively. There are a total of 64 data points, from 4 different games: Alien, Breakout, MsPacman, and Pong, where from 0-15 are from Alien, 16-31 are from Breakout and so forth.

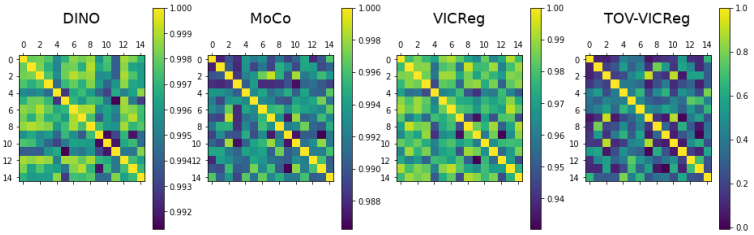

Figure 5: Similarity matrices of the representations computed by MoCo, DINO, VICReg, and TOV-VICReg respectively, of observations from MsPacman.

Attention visualisation 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 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, 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.

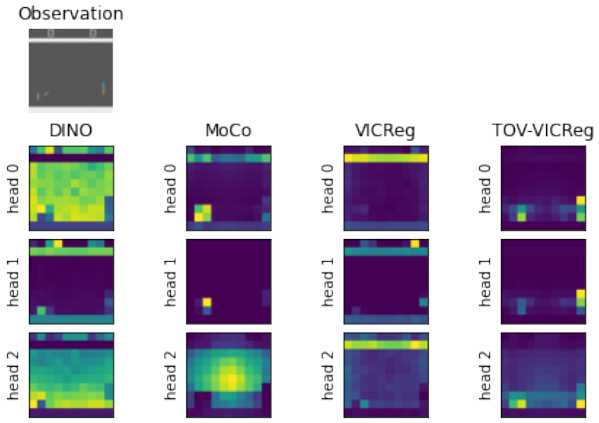

Figure 6: Attention maps produced by the pretrained ViTs. We fed a pretrained ViT with an observation from the game Pong and obtained the attention maps from the three heads in the last block.

9 Discussion & Conclusion

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 manages 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. These results reinforce the importance of encoding temporal relations between observations in the representation model, as shown by previous works, and also show that even vision models with weaker inductive biases and more parameters, when well pretrained, can achieve similar results in sample efficiency.

The ability to use larger models, with millions of parameters, that are as sample efficient as some of the most popular CNN-based models (like Nature CNN or Impala ResNet), with thousands of parameters, is very important since it opens the door to using Deep RL in even more complex problems where smaller models tend to struggle to perform, without losing sample-efficiency. Moreover, recent work in natural language processing (Devlin et al., 2019; Brown et al., 2020), and computer vision (Radford et al., 2021), shows great benefits from pre-training large models, and similar approaches in RL have the potential to unlock new levels of performance never achieved before (Baker et al., 2022).

References

Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. An Optimistic Perspective on Offline Reinforcement Learning. *arXiv:1907.04543*, June 2020.

Rishabh Agarwal, Marlos C. Machado, Pablo Samuel Castro, and Marc G. Bellemare. Contrastive Behavioral Similarity Embeddings for Generalization in Reinforcement Learning. *arXiv:2101.05265*, March 2021a.

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C Courville, and Marc Bellemare. Deep Reinforcement Learning at the Edge of the Statistical Precipice. In *Advances in neural information processing systems*, 2021b.

Ankesh Anand, Evan Racah, Sherjil Ozair, Yoshua Bengio, Marc-Alexandre Côté, and R. Devon Hjelm. Unsupervised State Representation Learning in Atari. Technical Report arXiv:1906.08226, arXiv, November 2020.

Kai Arulkumaran. Rainbow, August 2022. URL `https://github.com/Kaixhin/Rainbow`.

Bowen Baker, Ilge Akkaya, Peter Zhokhov, Joost Huizinga, Jie Tang, Adrien Ecoffet, Brandon Houghton, Raul Sampedro, and Jeff Clune. Video PreTraining (VPT): Learning to Act by Watching Unlabeled Online Videos. *arXiv:2206.11795*, June 2022.

Adrien Bardes, Jean Ponce, and Yann Lecun. Vicreg: Variance-invariance-covariance regularization for self-supervised learning. In *ICLR 2022-10th International Conference on Learning Representations*, 2022.

Marc G. Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 449–458, 06–11 Aug 2017.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *arXiv:1606.01540*, June 2016.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901, 2020.

Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. Unsupervised Learning of Visual Features by Contrasting Cluster Assignments. In *Advances in Neural Information Processing Systems*, volume 33, pp. 9912–9924, 2020.

Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging Properties in Self-Supervised Vision Transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 9650–9660, 2021.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A Simple Framework for Contrastive Learning of Visual Representations. In *Proceedings of the 37th International Conference on Machine Learning*, pp. 1597–1607, November 2020a.

Xinlei Chen, Haoqi Fan, Ross Girshick, and Kaiming He. Improved Baselines with Momentum Contrastive Learning. *arXiv:2003.04297*, March 2020b.

Xinlei Chen, Saining Xie, and Kaiming He. An Empirical Study of Training Self-Supervised Vision Transformers. *arXiv:2104.02057*, August 2021.

Stéphane d'Ascoli, Hugo Touvron, Matthew Leavitt, Ari Morcos, Giulio Biroli, and Levent Sagun. ConViT: Improving Vision Transformers with Soft Convolutional Inductive Biases. *arXiv:2103.10697*, June 2021.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv:1810.04805*, May 2019.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *International Conference on Learning Representations*, September 2020.

Dumitru Erhan, Aaron Courville, Yoshua Bengio, and Pascal Vincent. Why Does Unsupervised Pre-training Help Deep Learning? In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pp. 201–208, March 2010.

Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Matteo Hessel, Ian Osband, Alex Graves, Volodymyr Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, Charles Blundell, and Shane Legg. Noisy Networks For Exploration. February 2018.

Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H. Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, Bilal Piot, Koray Kavukcuoglu, Rémi Munos, and Michal Valko. Bootstrap your own latent: A new approach to self-supervised Learning. *arXiv:2006.07733*, September 2020.

Daniel Guo, Bernardo Avila Pires, Bilal Piot, Jean-bastien Grill, Florent Altché, Rémi Munos, and Mohammad Gheshlaghi Azar. Bootstrap Latent-Predictive Representations for Multitask Reinforcement Learning. *arXiv:2004.14646*, April 2020.

Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum Contrast for Unsupervised Visual Representation Learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9729–9738, 2020.

Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining Improvements in Deep Reinforcement Learning. *arXiv:1710.02298*, October 2017.

Ilya Kostrikov, Denis Yarats, and Rob Fergus. Image Augmentation Is All You Need: Regularizing Deep Reinforcement Learning from Pixels. *arXiv:2004.13649*, March 2021.

Misha Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. Reinforcement Learning with Augmented Data. In *Advances in Neural Information Processing Systems*, volume 33, pp. 19884–19895, 2020.

Xiang Li, Jinghuan Shang, Srijan Das, and Michael S. Ryoo. Does Self-supervised Learning Really Improve Reinforcement Learning from Pixels? *arXiv:2206.05266*, June 2022.

Ishan Misra, C. Lawrence Zitnick, and Martial Hebert. Shuffle and Learn: Unsupervised Learning Using Temporal Order Verification. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling (eds.), *Computer Vision – ECCV 2016*, Lecture Notes in Computer Science, pp. 527–544, Cham, 2016.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning. *arXiv:1312.5602*, December 2013.

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

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning Transferable Visual Models From Natural Language Supervision. In *Proceedings of the 38th International Conference on Machine Learning*, pp. 8748–8763, July 2021.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized Experience Replay. In *ICLR (Poster)*, January 2016.

Max Schwarzer, Ankesh Anand, Rishab Goel, R. Devon Hjelm, Aaron Courville, and Philip Bachman. Data-Efficient Reinforcement Learning with Self-Predictive Representations. *arXiv:2007.05929*, May 2021a.

Max Schwarzer, Nitarshan Rajkumar, Michael Noukhovitch, Ankesh Anand, Laurent Charlin, R Devon Hjelm, Philip Bachman, and Aaron C Courville. Pretraining Representations for Data-Efficient Reinforcement Learning. In *Advances in Neural Information Processing Systems*, volume 34, pp. 12686–12699, 2021b.

Aravind Srinivas, Michael Laskin, and Pieter Abbeel. CURL: Contrastive Unsupervised Representations for Reinforcement Learning. *arXiv:2004.04136*, September 2020.

Adam Stooke, Kimin Lee, Pieter Abbeel, and Michael Laskin. Decoupling Representation Learning from Reinforcement Learning. In *Proceedings of the 38th International Conference on Machine Learning*, pp. 9870–9879, July 2021.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning, second edition: An Introduction*. November 2018.

Tianxin Tao, Daniele Reda, and Michiel van de Panne. Evaluating Vision Transformer Methods for Deep Reinforcement Learning from Pixels. *arXiv:2204.04905*, May 2022.

Hado van Hasselt, Arthur Guez, and David Silver. Deep Reinforcement Learning with Double Q-Learning. In *Thirtieth AAAI Conference on Artificial Intelligence*, March 2016.

Hado P van Hasselt, Matteo Hessel, and John Aslanides. When to use parametric models in reinforcement learning? In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Han Wang, Erfan Miahi, Martha White, Marlos C. Machado, Zaheer Abbas, Raksha Kumaraswamy, Vincent Liu, and Adam White. Investigating the Properties of Neural Network Representations in Reinforcement Learning. *arXiv:2203.15955*, March 2022.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling Network Architectures for Deep Reinforcement Learning. In *Proceedings of The 33rd International Conference on Machine Learning*, pp. 1995–2003, June 2016.

Denis Yarats, Rob Fergus, Alessandro Lazaric, and Lerrel Pinto. Reinforcement Learning with Prototypical Representations. In *Proceedings of the 38th International Conference on Machine Learning*, pp. 11920–11931, July 2021.

Albert Zhan, Philip Zhao, Lerrel Pinto, Pieter Abbeel, and Michael Laskin. A Framework for Efficient Robotic Manipulation. *arXiv:2012.07975*, December 2020.

A ViT path size study

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 3: Scores obtained by training Rainbow, using different path sizes for ViT, in MsPacman for 100k steps across 10 different seeds

B 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

    # loss
    loss = inv_coef * inv_loss + var_coef * var_loss + cov_coef *
   cov_loss + temp_coef * temp_loss

    # optimization step
    loss.backward()
    optimizer.step()
```

Listing 1: Pytorch-like TOV-VICReg pseudocode

C TOV-VICReg augmentations

```
# Augmentation 1 / tau
RandomResizedCrop(84, scale=(0.08, 1.)),
RandomApply([
    ColorJitter(0.4, 0.4, 0.2, 0.1)
], p=0.8),
RandomGrayscale(p=0.2),
RandomApply([GaussianBlur((7, 7), sigma=(.1, .2))], p=1.0),
```

```
RandomHorizontalFlip()

# Augmentation 2 / tau prime
RandomResizedCrop(84, scale=(0.08, 1.)),
RandomApply([
    ColorJitter(0.4, 0.4, 0.2, 0.1)
], p=0.8),
RandomGrayscale(p=0.2),
RandomApply([GaussianBlur((7, 7), sigma=(.1, .2))], p=0.1),
RandomSolarize(120, p=0.2),
RandomHorizontalFlip(),

# Augmentation 3 / tau two prime and tau three prime
RandomApply([
    ColorJitter(0.4, 0.4, 0.2, 0.1)
], p=0.8),
RandomGrayscale(p=0.2),
```

Listing 2: Pytorch-like pseudocode of TOV-VICReg augmentations

D Rainbow implementation

We trained our agents using a PyTorch implementation of the Rainbow algorithm available on GitHub (Arulkumaran, 2022), which offers enough flexibility to adapt it to our needs. In Table 4 we present a comparison between the implementation used and the official results reported by DER (van Hasselt et al., 2019), 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 (Brockman et al., 2016), where for all games we used version number four of the environments (v4), disabled the default frame skip, and wrapped it with the DQN wrappers.

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 4: Comparison between DER scores and our implementation scores

E Atari Environments setup

We used the Atari games available at the gym library (Brockman et al., 2016) (version 0.23.1), and all games were run using their 4th version without frame skip, e.g. "AlienNoFrameskip-v4". Furthermore, we employ similar wrappers to the environments as previous works (Mnih et al., 2015), namely, scale observation to 84x84, change observations to grayscale, stack observations, apply a max number of no-op actions, and terminate the environment when the agent loses a life.

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

Listing 3: Gym Atari Wrappers

F 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 5: 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 6: MoCo v3 hyperparameters

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

Table 7: VICReg hyperparameters

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

Table 8: TOV-VICReg hyperparameters

G Models used

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

Table 9: Number of learnable parameters of each model we used

Games	Nature CNN	SGI ResNet-L	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 10: Table of results (mean and standard error) from experiments presented in Section 6.The bold values represent the best scores for the corresponding game

H Results Table

I Data-efficiency in unseen environments

Table 11 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.

Games	ViT tiny	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

Table 11: 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.