A DQN ALGORITHM PSEUDOCODE

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 $\mathcal{D}$;
    $y_j = r_j + \gamma \ \ max_{a'_j} Q_\phi(s'_j, a'_j)$;
    $\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi(s_j, a_j)}{d\phi}(Q_\phi(s_j, a_j) - y_j)$;
  end
end

Algorithm 1: DQN algorithm

B 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 1: Scores obtained by training Rainbow, using different path sizes for ViT, in MsPacman for 100k steps across 10 different seeds

C 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 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.

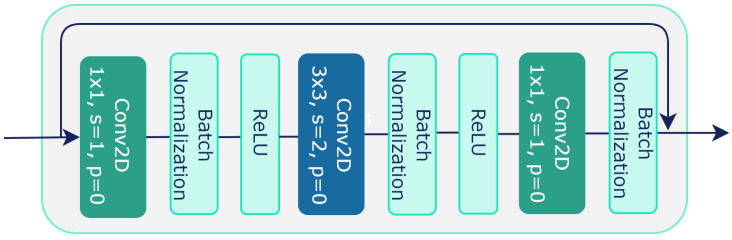

Figure 1: ResNet residual block

D 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

E 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

F RAINBOW IMPLEMENTATION

We trained our agents using a PyTorch implementation of the Rainbow algorithm available on GitHub, which offers enough flexibility to adapt it to our needs. In Table 2 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 2: Comparison between DER scores and our implementation scores

G 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

H 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 3: 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 4: 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 5: 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 6: TOV-VICReg hyperparameters

I MODELS USED

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

Table 7: Number of learnable parameters of each model we used

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 8: Table of results (mean and standard error) from experiments presented in Section 6.The bold values represent the best scores for the corresponding game

J RESULTS TABLE

K Data-efficiency in unseen environments

Table 9 shows a comparison of the randomly initialized and a 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 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 9: 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.

L Probability of improvement

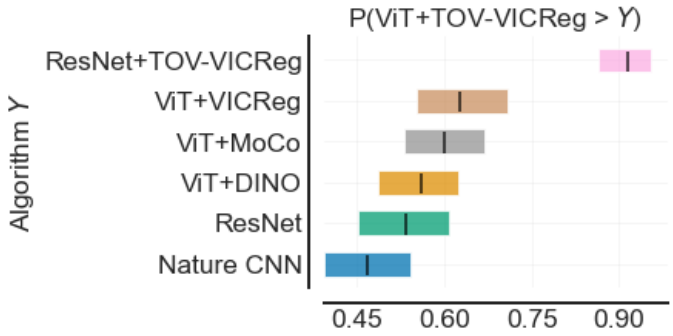

Figure 2: The probability of the Rainbow agent using the ViT pretrained with TOV-VICReg being better than the remaining agents

M 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 Anand et al. (2020), 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). 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

		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 10: 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.

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 10, 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 0.5276. 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 3. 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.

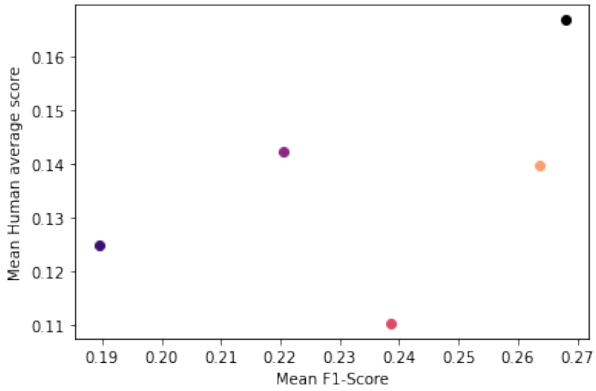

Figure 3: Relation between the mean average human score obtained in RL and the mean F1-score obtained in the evaluation task of several experiments