Learning Representations for Atari with Improved EfficientZero

Stanford CS 231N: Deep Learning for Computer Vision

Matthew Harvill
Stanford University
mharvill@stanford.edu

Abstract

In this project we implement and compare existing and novel variants of EfficientZero [14] on the Atari 100k benchmark to improve state representations and our agents' in-game performance. We test methods including using data augmentation, self-supervised consistency objectives, and masked reconstruction objectives. From our experiments, we find that state of the art masking methods for Model-free Reinforcement Learning (RL) algorithms do not improve performance with EfficientZero's model-based self-supervised consistency loss. We also confirm the importance of data augmentation in the constrained Atari 100k setting, seeing that agents trained without augmented observations fail to learn. Our findings suggest that the self-supervision available to Model-based RL algorithms significantly outperforms other contrastive or masked reconstruction based self-supervised learning methods, and that there is still much progress to be made in sample efficient state representation learning.

1. Introduction

The goal of this project is to improve our RL agents' performance on The Atari 100k Benchmark [7] with richer state representations. The Atari 100k Benchmark is a popular RL benchmark for comparing algorithms' sample efficiency. Due to the challenge of only interacting with the environment 100k times, learning expressive state representations is vital to the success of the RL agent(s). As such, it has been the test bed of a variety of self-supervised representation learning methods, including contrastive methods [13], data augmentation methods [8], and masking methods [15]. Although these representation learning methods continue to improve, they are generally evaluated on simple model-free RL algorithms for comparison. EfficientZero [14], the starting point for our project, takes a different approach.

EfficientZero attacks Atari 100k with the sole goal of maximal performance. It combines representation learning methods with a learned model of the environment borrowed from [10] to significantly outperform all other attempts at Atari 100k. Concretely, EfficientZero's representation and dynamics network architectures are ResNet-based [6], and they are jointly trained end-to-end with a self-supervised consistency loss function inspired by [2]. In this project we train EfficientZero on Breakout and MsPacman, two popular Atari games, with the following configurations:

- Raw observation inputs trained with base EfficientZero
- Augmented observations (random shifts and intensity alterations) trained with base EfficientZero
- Augmented Masked observations trained with modified EfficientZero
- Augmented Temporal Masked observations trained with modified EfficientZero according to [15]
- Augmented Selective Temporal Masked observations (Breakout only) with modified EfficientZero trained according to [15]

We then measure the success of our methods' state representations in terms of their downstream in-game performance on Breakout and MsPacman. We find that EfficientZero's base self-supervised consistency objective outperforms all of our other methods.

2. Related Work

In this section we dive deeper into the details about recent methods showcased on Atari 100k. Specifically, a combination of data augmentation and masking has shown to outperform contrastive approaches on Atari 100k.

2.1. Contrastive Representation Learning

The contrastive learning objective is to maximize the similarity between an anchor and positive examples and minimize the similarity between the anchor and negatives, and is widely used to construct better image representations [1] [5]. Contrastive Unsupervised Representations for Reinforcement Learning (CURL) [13] is a constrastive representation learning method on top of underlying RL algorithms that has achieved state of the art results on Atari 100k. CURL adds an auxiliary loss that is based on a bilinear inner-product similarity measure between anchor, positive, and negative image observations, where positive examples are constructed from random crops of the current observation and negatives are taken from different observations. Unlike other contrastive methods in RL, the authors attribute CURL's success to its focus on the current observation instead of attempting to also predict future representations like in [11].

2.2. Data Augmentation

Data augmentation is widely used in computer vision (CV) because it is simple and effective. However, common transformations like flips/rotations can not be applied to image-based RL, because they alter image semantics. The authors of Data-regularized Q (DrQ) [8] are the first to demonstrate that data augmentation can be used in RL in the form of random shifts. Specifically, DrQ uses data augmentation to regularize Q-value estimates by averaging over K image transformations in the calculation of the Q-target and averaging over M transformations in the standard l_2 loss based Q-function update. This simple method outperforms other representation learning methods like CURL without the need for auxiliary objectives or algorithmic alterations.

2.3. Mask-based Representation Learning

Mask-based Latent Reconstruction (MLR) [15] is an auxiliary objective optimized with policy learning objectives that draws inspiration from the success of mask-based pretraining in natural language processing (NLP) and CV [3]. Just like in CV, MLR masks original observations in the pixel space. However, instead of learning how to reconstruct the pixels, MLR first encodes the observations with a CNN-based Encoder and reconstructs masked pixels in the latent space, since pixels often contain distractions or redundancies for learning effective policies. The reconstruction is done with a Transformer Encoder that also uses actions as inputs, along with positional embeddings since temporal information is important. The reconstructed latent representations are then optimized with a cosine similarity loss to match the original image representations, similar to [4].

3. Methods

3.1. Baseline (with Augmentation)

Our baseline method for training effective state represetations from Atari image observations is summarized by the right half of 1, specifically with respect to the consistency loss, $L_{consistency}$. Training of this method is performed on two observations, o_t and o_{t+1}. First, these image observations are augmented with a random shift of 0-4 pixels in both horizontal and vertical directions. Then their intensity is varied by $\pm 5\%$. Let's denote these augmented observations $a(o_t)$ and $a(o_{t+1})$. After augmenting the observations, both $a(o_t)$ and $a(o_{t+1})$ are fed as inputs into EfficientZero's representation network. The representation network is a ResNet, with convolutional layers, batch normalization layers, pooling, ReLU nonlinearities, and residual connections. The outputs of this network are the state representations for the original o_t and o_{t+1}, which we will denote s_t and s_{t+1}.

Since EfficientZero is model-based, s_t and s_{t+1} can be used to train the dynamics function (predicts the next state given the current state and actions). This is done by feeding s_t and the action taken in this state, a_t, into the dynamics network, which is also a ResNet. The dynamics network outputs $\hat{s}_{t+1}$, a prediction of the next state s_{t+1}, which we have already generated from our training data. In order to compare s_{t+1} with $\hat{s}_{t+1}$ we first project them with a shared linear transformation, then apply a prediction layer on s_{t+1}'s projection, and finally compute the cosine similarity between these projections to attain our loss, $L_{consistency}$. Notice that we apply a stop gradient to s_{t+1}'s projection. This is to prevent our model from collapsing (e.g. learning the same representation for every state). Notice that we could also simply compute the cosine similarity loss with respect to s_{t+1} and $\hat{s}_{t+1}$, but empirically this is shown to degrade state representations, possibly because the representation network removes important information in order to bring s_{t+1} and $\hat{s}_{t+1}$ closer together.

3.2. Baseline without Augmentation

We also experiment with the base EfficientZero architecture using raw image observations. This method follows the same process as 3.1, except we pass o_t and o_{t+1} directly as inputs into the representation network.

3.3. Masked Latent Reconstruction

Our masked latent reconstruction method builds on 3.1 and is summarized in 1. While $L_{consistency}$ is used to train our representation and dynamics networks end-to-end, L_{MLR} only trains our representation network end-to-end with a throwaway decoder network that is only used for our auxiliary masked latent reconstruction task. For this task, we start with two copies of our original o_t and augment them like before to obtain two copies of $a(o_t)$. Then we apply a mask to the right copy, where 50% of the pixels are zeroed out in the form of uniformly randomly selected 12x12 chunks (o_t and $a(o_t)$ are 96x96 pixels). A preview of this can be seen in 2*c* and 3*c*. We will call this masked observation $m(a(o_t))$. After generating $m(a(o_t))$, we pass

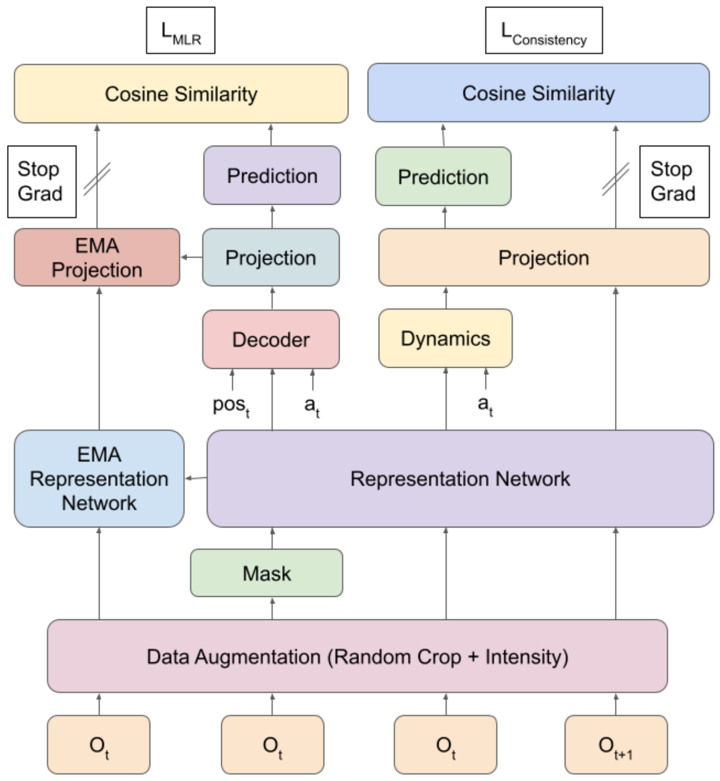


Figure 1. Self-Supervised Loss Architecture with Masked Latent Reconstruction

$a(o_t)$ and $m(a(o_t))$ through two representation networks, where our the leftmost network shown in 1 is an exponentially weighted average of our true representation network. This idea comes from [5] and was also found to work better than a copy of the original network in [15] since it gives the target output more information of past observations, and thus we have also opted to use it. We will denote the output of the EMA representation network $\overline{s}_t$ and the output of the regular network (from a masked input) as s_t^m.

Since the goal of this method is to use masking to train the representation network to output more robust, context-dependent state representations, we use a Transformer Encoder as our decoder network to transform s_t^m into $\hat{\hat{s}}_t$, a prediction of the non-masked o_t's representation. Our decoder network is made up of two transformer encoder blocks and takes s_t^m and a_t as input to generate $\hat{\hat{s}}_t$. Specifically, a_t in this context is a learned embedding for the action taken in this state that has the same dimension as s_t^m. For this method (our most basic masking method), we only compute self-attention between s_t^m and a_t, without the need for positional embeddings.

Next, similar to how we add projection and prediction heads in our self-supervised consistency loss, we also transform $\overline{s}_t$ and $\hat{\hat{s}}_t$ with projection and prediction heads, where the projection head for $\overline{s}_t$ is an exponential moving average of the projection head for $\hat{\hat{s}}_t$ for the same reason we use an EMA-based representation network. We then compute the cosine similarity as we do for the consistency loss, adding a stop gradient to the target projection to prevent model collapse to arrive at L_{MLR}. This method presents a self-supervised auxiliary loss that we combine with EfficientZero's self-supervised loss with the goal of improving our representation network's ability to represent salient features in the input observations.

3.4. Temporal Masked Latent Reconstruction

For temporal masked latent reconstruction, we make a slight change to 3.3 by processing multiple observations at once. Specifically we pass $a(o_t), a(o_{t+1}), ..., a(o_{t+4})$ as inputs into our EMA representation network and $m(a(o_t)), m(a(o_{t+1})), ..., m(a(o_{t+4}))$ as inputs into our representation network. When combined with our action embeddings $a_t, a_{t+1}, ..., a_{t+4}$ and positional embeddings, $p_t, p_{t+1}, ..., p_{t+4}$, we are able to perform temporal self-attention on all of $\hat{\hat{s}}_t, \hat{\hat{s}}_{t+1}, ..., \hat{\hat{s}}_{t+4}$ with their respective actions. Concretely, the input to our decoder network is the sequence, $a_t + p_t, \hat{\hat{s}}_t + p_t, a_{t+1} + p_{t+1}, \hat{\hat{s}}_{t+1} + p_{t+1}, ..., a_{t+4} + p_{t+4}, \hat{\hat{s}}_{t+4} + p_{t+4}$. We then compute the projections and loss as before, summing over time instead of over a single timestep, t.

3.5. Selective Temporal MLR

For our final method, we use the same architecture and process as 3.4, only we change how the observations are masked. Specifically, we add this method for Breakout because we believe that the previous masking strategy is too destructive, reducing meaningful training signals. For this method we either keep the score/lives info section of the image or the bricks section with a 50% probability assigned to either outcome. These two masks are shown in 2d and 2e.

3.6. Code - My Contributions

For this project I use EfficientZero [14] and MLR's [15] open-source code as a starting point. My contributions for this project are:

- Training existing EfficientZero implementations with and without data augmentation for Breakout and MsPacman
- Incorporating MLR's auxiliary loss into EfficientZero and training it on Breakout and MsPacman
- Adding brick and score info masks to the temporal MLR architecture for Breakout

4. Dataset and Features

The data for this project is collected during the RL agent's online exploration. Therefore, it is a variable dataset. Given that we train each model for 50k timesteps, there are 50k training observations for each agent for both Breakout and MsPacman. The observations are obtained from the Gymnasium environment simulator and are reshaped into (96x96x3) images, corresponding to 96x96 pixels with RGB channels for each pixel. These observations are shown in 2a and 3a. Instead of extracting features, the raw pixel data is passed into our models, but as we mention in 3, we also perform augmentations and masking. An example of augmentation for Breakout is shown in 2b and for MsPacman is shown in 3b. At first glance it is difficult to tell the difference between these images, but this is actually the point. These small augmentations retain the semantics of each observation, while providing much more varied inputs to the representation network, allowing it to learn to generalize.

We also illustrate examples of our masking approaches. The masking borrowed from [15] is shown in 2c and 3c. From these depictions we notice that most of a Breakout observation is already black (zeroed out), and so most of the mask has no effect. Additionally, we also notice that masking cuts out a lot of information that might not be recoverable given the remaining image pixels (e.g. the agent's cursor, the number of remaining lives). For these reasons, we also experiment with our selective masking (keeping the score/lives info or keeping the bricks, each with 50% probability) and only masking other parts of the input observations. These configurations are shown in 2d and 2e. Our idea is that the network can learn to reconstruct some of the score info from the brick configuration or the brick configuration from the score info. We do not experiment with another masking structure for MsPacman because MsPacman has a richer context to infer from, and therefore we believe masking will not be as potentially damaging.

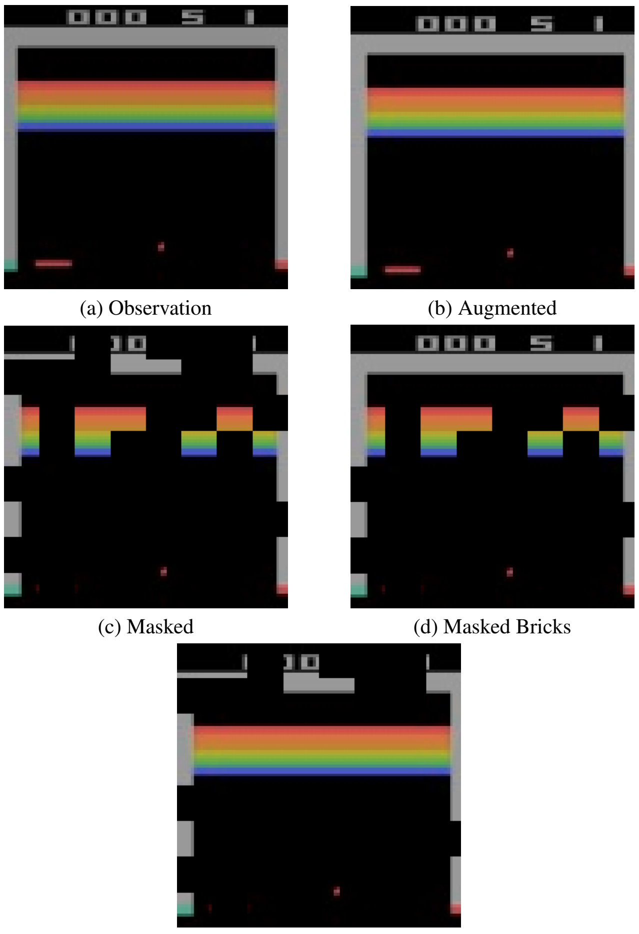

(a) Observation (b) Augmented (c) Masked (d) Masked Bricks (e) Masked Info

Figure 2. Breakout observation images before and after transformations/masking.

5. Results and Discussion

5.1. Experimental Details

Since the process of training an RL agent with MCTS as the policy improvement operator as done in AlphaZero-

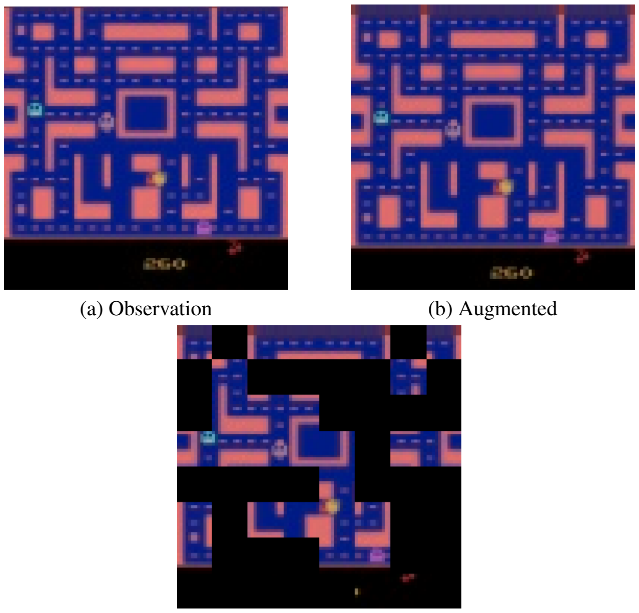

(a) Observation (b) Augmented (c) Masked

Figure 3. MsPacman observation images before and after transformations/masking.

style algorithms [12] is computationally expensive, we do our best to simplify the number of variables involved in our experiments. Additionally, since EfficientZero [14] and Masked Latent Reconstruction [15] boast SOTA performance, we choose to reuse their hyperparameters and architectural choices for most of our methods. This includes using EfficientZero's ResNet representation network with ReLU activations, the same learning rate, and Stochastic Gradient Descent (SGD) as the optimizer. Even though we probably could obtain better performance with an adaptive optimizer like Adam or nonlinearities like GeLU, we want to be able to attribute our results solely based on our design decisions. This also includes using the same initial masking procedure as the authors of MLR [15], the same decoder network (Transformer Encoder with two attention layers, and a single attention head), and the same positional embeddings. Our main decisions are choosing the selective mask for Breakout and choosing the loss ratio for our auxiliary masked reconstruction loss, which we set equal to the weightage as the other loss terms (value loss, policy loss etc.) except for the consistency loss, which has an upweight of 2. Given that the consistency loss trains both the dynamics and representation networks, we believe it deserves higher weightage than the MLR loss.

5.2. Breakout Results

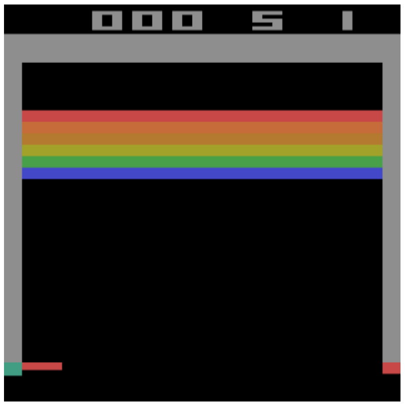
We implement five methods for Breakout and four methods for MsPacman. For both games these methods include using raw images with EfficientZero, random shift and intensity augmented images, augmented masked images trained with additional reconstruction loss, and a temporal masking implementation. For breakout we additionally train with our selective masking. The quantitative results from these methods are shown in 5 and 6, with our Selective Masking method's results omitted since they are almost identical to the Temporal MLR results. This also makes it easier to compare these methods across Breakout and MsPacman

What we immediately notice is that pure EfficientZero with augmentations outperforms other methods, the masking methods perform relatively similarly, and that the agent with raw observation inputs fails on both games. For breakout specifically, we notice that the mean score of the fully trained augmentation agent is over double that of the agents trained with the auxiliary MLR loss. Since the unweighted MLR loss for both implementations is similar to the unweighted consistency loss (both converge), this leads us to believe that the masked objective is damaging to the representations for Breakout. This confirms some of our ideas that the masking removes information that can not be reconstructed from the remaining pixels. However, it is interesting that with our selective masking, we also obtain similar results, since our new masking seems like it addresses some of the drawbacks of our other masking patterns, notably that it does not force the model to predict parts of the state that are impossible to predict from the remaining information. Given that this method still achieves suboptimal performance, we believe that learning to reconstruct the image forces the representation to generalize too much and that intricacies between different states are actually very important to the agent's success.

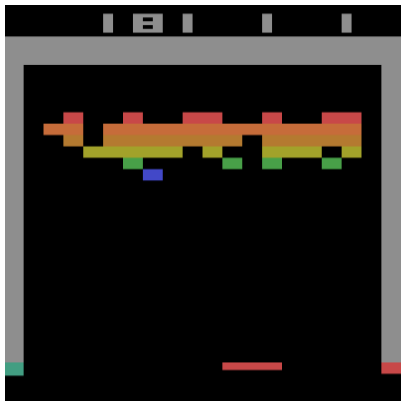
After viewing the agent's performance, we are led to the same conclusion - that similar observations should not be grouped/generalized. Since the user's paddle is surprisingly sensitive, small changes in its position lead to drastically different ball reflection trajectories. Being able to control these slight differences proves to be vital to achieving higher scores. Although it is not as evident from screenshots, we can still begin to see this pattern when comparing 4*b* and 4*c*. The agents obtain much higher returns if they can create tunnels on the outside that lead to breaking bricks on the upper layers. This strategy is only learned by the agent without masking, which makes sense since the masking objective appears to overly generalize, and thus the masking agents only learn to deflect the ball, rather than being able to control the ball's new path more consistently.

5.3. MsPacman Results

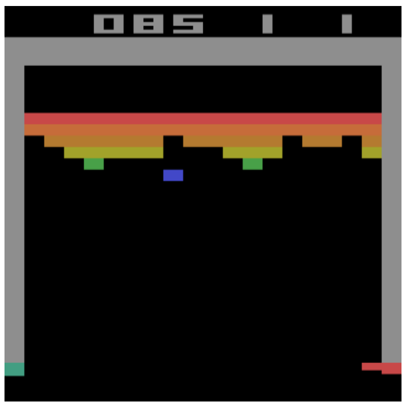
For MsPacman, we also notice that the augmentation-only method with EfficientZero's self-supervised consistency loss outperforms our masking methods, although the margin is smaller. We think this is because the masking



(a) No Augmentation



(b) Augmentation



(c) (Temporal) MLR

Figure 4. Average (typical) agent performance on Breakout across representation learning methods

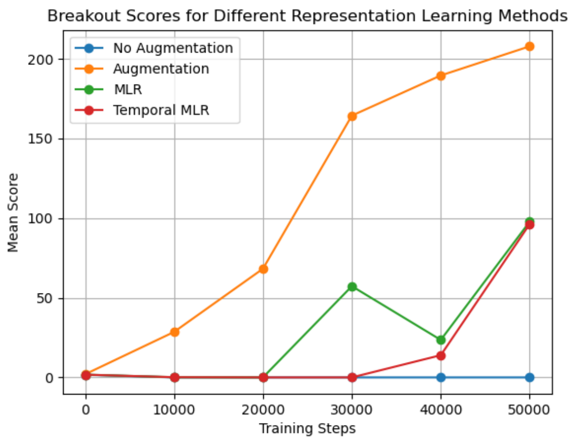


Figure 5. Mean score on Breakout across representation learning methods for 50k environment interactions.

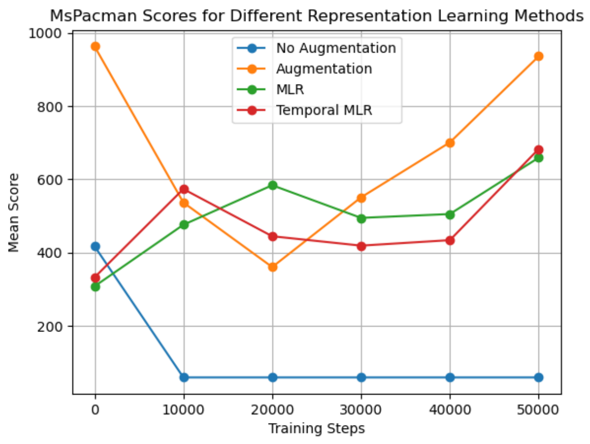


Figure 6. Mean score on MsPacman across representation learning methods for 50k environment interactions.

of MsPacman pixels is less detrimental to the context, and thus the representation network can still learn how to effectively fill in gaps from the remaining pixels. However, its decreased performance still suggests that certain intricacies of the input are lost when the masking objective is added, similar to our Breakout agents.

Another complication with MsPacman is that it is a challenging game. Our inspection confirms this as the results seem much more random compared to Breakout. After watching videos of agent gameplay, it does not seem like any of the agents have clear strategies (which is why we do not include screenshots from MsPacman gameplay). Although it is difficult to tell if agents are learning to play MsPacman, mean performance is still indicative of success as the mean scores are computed over 32 agents.

5.4. MLR vs MLR + Consistency

Although we have multiple hypotheses about why our masking methods perform worse than augmentation-only, we believe the main reason is that masked latent reconstruction is a weaker self-supervision method than the consistency objective and that it interferes with the consistency updates. Since EfficientZero learns a dynamics function, its self-supervised consistency loss trains both the representations and the dynamics network, and the modeled transitions inherently embeds a lot of information into the state representations. However, without this convenient learned model of the environment for self-supervision, MLR is unable to grasp the return-based aspect of the representations. With this in mind, it makes sense that MLR achieves SOTA performance on Model-free RL methods, yet fails to improve the performance of EfficientZero.

5.5. No Augmentation

We are surprised that the agents trained without data augmentation are not able to learn how to play effectively at all. However, given that RL algorithms such as Deep Q Networks (DQN) [9] are trained on millions of interactions in order to learn, it makes sense that without data augmentation, our agent does not have enough data to learn effective policies. Our findings emphasize how important data augmentation is in sample-constrained settings.

6. Conclusion and Future Work

For this project we focus mainly on data augmentation methods and self-supervised representation learning. We find that data augmentation is vital to performance on the Atari 100k benchmark and that EfficientZero's self-supervised consistency loss results in better state representations compared to masked reconstruction based self-supervised state representation learning methods. This might seem surprising at first, but after further thought, we believe it makes a lot of sense due to the rich guidance of EfficientZero's dynamics model.

Given more time and compute, we would like to explore more masking patterns before conceding that masked reconstruction is only suited for model-free RL systems. We would also like to train each model for the full 100k timesteps, as we can't be entirely sure that our methods would still lag behind after training on more examples (this is particularly possible given temporal MLR's seemingly exponential growth in performance on Breakout towards the 50k mark).

7. Contributions and Acknowledgements

As this is a solo project, Matthew Harvill performed all parts of the project.

References

[1] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton. A simple framework for contrastive learning of visual representations. *CoRR*, abs/2002.05709, 2020. 2

[2] Xinlei Chen and Kaiming He. Exploring simple siamese representation learning. *CoRR*, abs/2011.10566, 2020. 1

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. 2

[4] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H. Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Ávila 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. *CoRR*, abs/2006.07733, 2020. 2

[5] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross B. Girshick. Momentum contrast for unsupervised visual representation learning. *CoRR*, abs/1911.05722, 2019. 2, 3

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. 1

[7] Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H. Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, Ryan Sepassi, George Tucker, and Henryk Michalewski. Model-based reinforcement learning for atari. *CoRR*, abs/1903.00374, 2019. 1

[8] Ilya Kostrikov, Denis Yarats, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. *CoRR*, abs/2004.13649, 2020. 1, 2

[9] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. 7

[10] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy P. Lillicrap, and David Silver. Mastering atari, go, chess and shogi by planning with a learned model. *CoRR*, abs/1911.08265, 2019. 1

[11] Max Schwarzer, Ankesh Anand, Rishab Goel, R. Devon Hjelm, Aaron C. Courville, and Philip Bachman. Data-efficient reinforcement learning with momentum predictive representations. *CoRR*, abs/2007.05929, 2020. 2

[12] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *CoRR*, abs/1712.01815, 2017. 5

[13] Aravind Srinivas, Michael Laskin, and Pieter Abbeel. CURL: contrastive unsupervised representations for reinforcement learning. *CoRR*, abs/2004.04136, 2020. 1, 2

[14] Weirui Ye, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, and Yang Gao. Mastering atari games with limited data. *CoRR*, abs/2111.00210, 2021. 1, 4, 5

[15] Tao Yu, Zhizheng Zhang, Cuiling Lan, Zhibo Chen, and Yan Lu. Mask-based latent reconstruction for reinforcement learning. *CoRR*, abs/2201.12096, 2022. 1, 2, 3, 4, 5