Low-Rank Tensor-Network Encodings for Video-to-Action Behavioral Cloning

Anonymous authors
Paper under double-blind review

Abstract

We describe a tensor-network latent-space encoding approach for increasing the scalability of behavioral cloning of a video game player's actions entirely from video streams of the gameplay. Specifically, we address challenges associated with the high computational requirements of traditional deep-learning based encoders such as variational auto-encoders that prohibit their use in widely available hardware or for large scale data. Our approach uses tensor networks instead of deep variational autoencoders for this purpose, and it yields significant speedups with no loss of accuracy. Empirical results on ATARI games demonstrate that our approach leads to a speedup in the time it takes to encode data and train a predictor using the encodings (between 2.6× to 9.6× compared to autoencoders or variational autoencoders). Furthermore, the tensor train encoding can be efficiently trained on CPU as well, which leads to comparable or better training times than the autoencoder and variational autoencoder trained on GPU (0.9× to 5.4× faster). These results suggest significant possibilities in mitigating the need for cost and time-intensive hardware for training deep-learning architectures for behavioral cloning.

1 Introduction

Learning by imitation is a paradigm that enables teaching complex tasks through observation of expert behavior (Hussein et al., 2017) including object manipulation (Shen et al., 2022), robotic locomotion (Wang et al., 2022), and autonomous driving (Bronstein et al., 2022). Several approaches for learning from data have been proposed, including behavioral cloning and inverse reinforcement learning.

In this work, we consider the task of learning directly from expert demonstrations in the form of action and video-frame sequences. In this context, learning directly from raw pixel data is computationally challenging due to the high resolution of game frames. This challenge is often addressed via downsampling and usage of large-scale computational resources. An illustrative example is the OpenAI VPT (Baker et al., 2022) for the game of Minecraft, which first downsamples the original frame size of 640×360 to 128×128. Furthermore, the model is trained using $70,000$ hours of gameplay, has half a billion parameters, and is trained for nine days on 720 V100 GPUs.

One approach to tackle the computational challenges of using raw video frames as input is to provide hand-tuned features (such as movement, location, terrain, visibility, etc.) rather than, or in addition to, raw pixel data. With this additional information, important elements of the game are provided and thus do not have to be learned solely from video data. This is the approach taken by some reinforcement learning and imitation learning works, such as OpenAI et al. (2019) for Dota 2 and Vinyals et al. (2019) for Starcraft II. However, such an approach requires human effort and insight in identifying traits that would be useful for a learning algorithm and developing a means for extracting this data. As an alternative to using hand-crafted features, there have been various approaches proposed for learning a latent space representation via self-supervised learning, where auxiliary tasks are used to train a neural network that maps high-dimensional input data into a lower dimensional space (Chen et al., 2021b). In this setting, autoencoder-based deep learning architectures have been a common method of choice for identifying this latent space (Jaques et al.,

2021; Brown et al., 2020; Chen et al., 2021a;b; Amini et al., 2018), although other approaches have also been explored (Sermanet et al., 2017; Pari et al., 2021; Chen et al., 2021b).

In this work, we replace latent space extraction via a more computationally efficient tensor network approach, based on the Tensor Train (TT) format (Oseledets, 2011). The TT-format offers a multi-linear approximation that can be tuned to have a target reconstruction accuracy without the need of extensive hyperparameter tuning. Furthermore, the recent development of an incremental algorithm, `TT-ICE`* (Aksoy et al., 2022), allows us to compute TT-decomposition of a large amount of video data faster and with less memory than the standard `TT-SVD` algorithm — a critical enabling innovation for this application. `TT-ICE`* computes the TT-decomposition entirely using basic linear algebra operations such as matrix multiplication and decomposition. Furthermore, it only requires a single pass through the data, a significant advantage over backpropagation techniques that can require multiple passes through the training data and do not guarantee any representation accuracies. The lack of guarantees on representation accuracy by deep-learning architectures also leads to the need for validation-data driven hyperparameter tuning to find hyperparameters and architecture settings that lead to high-quality representations. Instead, the complexity of the representation of our approach, represented by the tensor *rank*, can be guaranteed and found adaptively via the `TT-ICE`* algorithm.

While the TT-format may be viewed as having a smaller representational capacity compared to the nonlinear architecture of neural networks, there is evidence suggesting that neural networks might exhibit more expressiveness than required for specific tasks. Several works have found that some layers of trained neural networks can be replaced with tensor networks with little to no degradation of accuracy while improving the computational and memory costs of running these networks (Novikov et al., 2015; Garipov et al., 2016; Yu et al., 2017b; Kim et al., 2016; Lebedev et al., 2015; Sharma et al., 2021). In our work, we test whether encodings provided by neural networks can similarly be replaced by tensor networks. However, in contrast to the above works, we train the tensor network encodings directly, instead of first training a neural network encoding then replacing it (or components of it) with tensor networks.

In this paper, we demonstrate that this approach does provide a practical and highly effective alternative to deep-learning based encodings by reducing the requirements of hyperparameter tuning, thereby offering shorter training times for behavioral cloning. Moreover, the TT-encoding-based predictors achieve similar or higher behavioral cloning accuracies than predictors built on encodings from an convolutional autoencoder (AE) or variational autoencoder (VAE) (Kingma & Welling, 2014). Although other neural network approaches for feature extraction are available (see Chen et al. (2021b) for examples), we chose AEs and VAEs since like the TT, these are trained to minimize a loss that is related to reconstruction error.

To summarize, our contributions are as follows:

1. Empirical demonstration that the low dimensional embeddings obtained through `TT-ICE`* can be used for the downstream task of behavioral cloning.

2. Empirical demonstration of a reduction in training time ($2.6 - 9.6\times$ faster when trained on GPU), while achieving comparable behavioral cloning accuracy and gameplay scores compared to autoencoder-based approaches for extracting a latent space.

2 Related Work

In machine learning literature, AEs and VAEs are commonly used as feature extraction tools. Polic et al. (2019) uses a convolutional autoencoder (CAE) to extract features from images of a tactile sensor. The extracted features are then used to estimate contact object shape, edge position, orientation and indentation depth using shallow neural networks. Yu & Zhou (2020) extracts features from gearbox vibration data using CAE and uses the extracted features to identify fault modes. Pintelas et al. (2021) uses CAE to compress and de-noise high dimensional images to smaller size and then uses the encoded images for classification, Zabalza et al. (2016) uses a stacked AE to extract features from hyperspectral images and uses those features in classification, and Cai et al. (2022) uses AE-extracted-features to determine the level of traffic congestion from video frames.

There is also increased attention to transformer based methods. Yang et al. (2022) uses a transformer encoder to generate pair-wise relations for the objects in the image and uses a transformer decoder to generate captions. Kim et al. (2021) uses a transformer based architecture as feature extraction step in dual arm robot manipulation. However, training complex architectures like transformers increases the computational cost and data requirements even further. In Kim et al. (2021), it is reported that 8 NVIDIA V100 GPUs were used in training.

In addition to neural network-based tools, tensor methods have also been considered as a feature extraction option in machine learning literature. Tensor methods allow the extraction of meaningful features using linear algebra operations, therefore eliminating the need for backpropagation. Taherisadr et al. (2019) uses CANDECOMP/PARAFAC (CP) format to reduce the size of EEG signals and increase the computational efficiency of a CNN-based seizure predictor. Zhang et al. (2017) use Tucker format to extract features from PCG signals and a support vector machine based classifier to identify anomalous heart sounds. Fonał & Zdunek (2019) use TT-format to extract features from various multidimensional datasets and a k-nearest neighbor classifier to perform multi-class-classification. Zhao et al. (2020) use a TT layer to extract features from high-dimensional video data before passing those features into a recurrent neural network for the task of video summarization.

Although tensor methods and neural networks are both capable of feature extraction, there is the question of whether the nonlinearity of neural networks can lead to features that are better suited for downstream tasks than the multilinear tensor methods. Previous studies indicate that the increased expressiveness of neural networks might be excessive for certain tasks and computationally expensive compared to similar tensor-based methods. Several works have explored compressing portions of large neural networks after training, including fully connected layers (Novikov et al., 2015) and convolutional layers (Garipov et al., 2016; Yu et al., 2017b; Kim et al., 2016; Lebedev et al., 2015; Sharma et al., 2021). The fine-tuning of large pre-trained neural networks can also be simplified by exploiting low rank structure; Hu et al. (2022) explores using low-rank matrices to update the weights of large language models when fine-tuning these models for downstream tasks. Other work has also explored using tensor operations directly within the network architecture itself during training as a means for parameter reduction and model compression; examples include Tensor Contraction Layers (Kossaifi et al., 2017), Tensor Regression Layers (Kossaifi et al., 2020), and Higher-Order Tensor RNNs (Yu et al., 2017a). The ability to replace portions of deep neural networks with tensor networks either during or after training suggests that alternative representations may exist that do not require as much computational power and time to train, while not sacrificing accuracy.

In this work, we explore using a tensor network in TT-format as one such alternative representation, where the TT is used to extract features from video frame data for behavioral cloning. In comparison to using neural network based encodings like convolutional AEs and VAEs, we show that using a TT leads to significantly faster training times without sacrificing behavioral cloning accuracy.

3 Methodology

The overarching objective of this work is to extract a *game-specific* playing strategy from video demonstrations of an experienced agent. In this section we first describe our learning framework; then we present the methods to extract a low-dimensional representation and discuss how TT-decomposition fits within this setting. Finally, we briefly present the action predictor that maps latent representations to actions.

The imitation learning framework we use is shown in Figure 1. It consists of two parts: an encoding transformation (labeled "Frame encoder" in Figure 1) and an action predictor ("Action predictor" in Figure 1). The encoding transformation serves to reduce the dimensionality of the space so that the action predictor can potentially be learned more efficiently. In this paper we aim to accelerate computation of the encoding transformation.

The encoder works by mapping frames of a video sequence into an L dimensional latent-space. We compare three such encoders in this work: two neural network-based encoders determined from an AE and VAE, and a low-rank encoder based on the TT-decomposition. The dimensionality of the latent space is typically determined by the formulation of the encoding transformation. For example, AEs and VAEs would treat

this as a hyperparameter over which to optimize, whereas our low-rank approach will adaptively determine this dimension to a user-specified error tolerance. We further note that each of the three methods outlined above also have an ability to reconstruct an image from the latent space, which potentially can be useful for investigating the latent space.

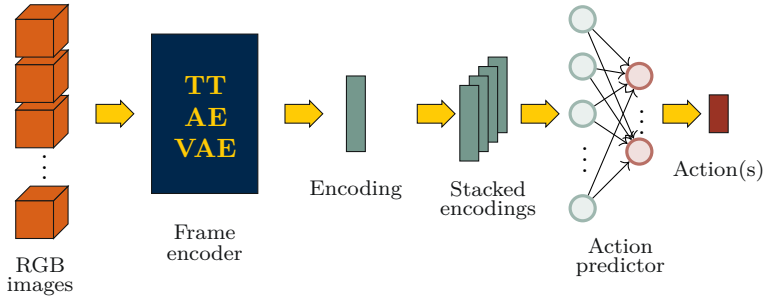


Figure 1: *Imitation learning framework. Our focus is to enhance latent space learning depicted by the blue box. We compare the TT, autoencoder (AE) and variational autoencoder (VAE) approaches. Orange cubes represent images from gameplay videos. The extracted latent features are then used in the action-prediction network to predict actions. The architecture of the action predictor remains the same for all three encoding methods.*

In the rest of this section, we will discuss the methods and tools to compute the latent space representation of the gameplay sequences efficiently in detail.

3.1 Frame Encoder

This section discusses three alternatives that we consider as frame encoding methods. We will first briefly introduce AE and VAE and then discuss our TT-based approach in detail.

As mentioned in Section 1, AEs and VAEs have found great use for extracting encodings in the context of imitation learning and behavioral cloning. However, low-rank tensor decompositions have not been widely used for this purpose.

3.1.1 Autoencoder

AEs are a neural network based latent space extraction tool. AEs are used for a variety of tasks, including dimensionality reduction(Wang et al., 2016), denoising(Majumdar, 2018), and image compression(Sreelakshmi & Ravi, 2020). AEs consist of two parts: an encoder and a decoder. The encoder takes the input data and maps it to a latent space representation. The decoder then takes the latent space representation and maps it back to the original data. The goal of the AE is to learn a mapping that minimizes the difference between the original data and the reconstructed data.

AEs are a powerful tool for unsupervised learning, and they can be used to extract meaningful features from data without any prior knowledge. However, AEs can be computationally expensive to train, and they can be sensitive to the choice of hyperparameters.

3.1.2 Variational Autoencoder

VAEs (Kingma & Welling, 2014) are similar to AEs in utilizing an encoding and decoding network. However, instead of mapping a fixed input to a fixed latent vector, the input is mapped probabilistically to a latent vector z. A benefit of VAEs is that it is a generative model, allowing one to generate new samples by sampling latent vectors z (typically) from a standard normal distribution and passing it through the decoder. An additional potential benefit is that the training of a VAE imposes some structure onto the latent space, as the latent space is trained to be normally distributed.

The AE and VAE are both implemented using an encoder consisting of four convolutional layers, followed by a flattening layer and a final linear layer, and a decoder consisting of a linear layer, a reshaping layer, and four transposed convolutional layers. Additional details on the architecture and training of the AEs and VAEs can be found in Appendix A.1.

3.1.3 Tensor-Train (TT) Format

The TT is a compact representation method for multi-dimensional arrays. A d-dimensional tensor $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$ is said to be in *TT-format* if it is represented as a product of d 3-dimensional tensors $\{\mathcal{G}_i\}_{i=1}^{d}$, $\mathcal{G}_i \in \mathbb{R}^{r_{i-1} \times n_i \times r_i}$, called *TT-cores*. The internal dimensions r_i of TT-cores $\mathcal{G}_i$ are also referred to as *TT-ranks* and are determined by error-truncated SVDs through a user-specified relative error tolerance $\varepsilon \in [0, 1]$, except for $r_0 = r_d = 1$.

In the case of RGB images, a video sequence is a 4-dimensional tensor $\mathcal{X} \in \mathbb{S}^{H \times W \times C \times N}$ with $\mathbb{S} = [0, 255]$. The first three dimensions correspond to the height (H), width (W), and color (C) channels; and the last dimension corresponds to the index of an individual image within the video sequence.

In the video application considered in this paper, we first reshape $\mathcal{X}$ into a 6-way tensor of shape $n_1 \times \ldots \times n_5 \times N$, where $\prod_{i=1}^{5} n_i = HWC$. By reshaping $\mathcal{X}$ to have smaller dimensions, we aim to limit the maximum TT-rank for each dimension. This also results in a slower growth in overall TT-ranks and yields higher compression. After reshaping, the tensor of a video sequence is represented in TT-format as

$$\mathcal{X} \approx \mathcal{G}_1 \ {}_3\times_1 \ \mathcal{G}_2 \ {}_3\times_1 \ \cdots \ {}_3\times_1 \ \mathcal{G}_6, \tag{1}$$

where $\mathcal{A} \ {}_3\times_1 \ \mathcal{B}$ denotes the tensor contraction between the third dimension of $\mathcal{A}$ and first dimension of $\mathcal{B}$, and $\mathcal{G}_i \in \mathbb{R}^{r_{i-1} \times n_i \times r_i}$.

Note that in Equation 1, the first 5 TT-cores, $\mathcal{G}_1, \ldots, \mathcal{G}_5$, correspond to reshapings of the H, W, and C of the images – these are common to all images. The final dimension indexes the unique frame and therefore only the final core is unique to a data point. To be more specific, for the i-th image stored in $\mathcal{X}$, we would have the reconstruction

$$\mathcal{X}(i) \approx \mathcal{G}_1 \ {}_3\times_1 \ \cdots \ {}_3\times_1 \ \mathcal{G}_5 \ {}_3\times_1 \ \mathcal{G}_6[:, i], \tag{2}$$

where $\mathcal{G}_6[:, i]$ denotes the i-th column of $\mathcal{G}_6$. As a result of this, we can use the representations stored in $\mathcal{G}_6$ as encodings of the frames and the fifth TT-rank, r_5, becomes the size of the latent space L.

One of the methods to compute a TT-decomposition for d-dimensional arrays is the `TT-SVD` algorithm (Oseledets, 2011, Alg. 1). This algorithm returns an approximation $\tilde{\mathcal{X}}$ that satisfies the inequality $\|\mathcal{X} - \tilde{\mathcal{X}}\|_F \leq \varepsilon_{des} \|\mathcal{X}\|_F$ for some predefined relative error upper bound ε_{des}. Moreover, when computed with `TT-SVD`, the first 5 TT-cores consist of reshapings of orthonormal vectors (i.e., left singular vectors). However, performing the `TT-SVD` on large tensors requires loading the full tensor into memory and performing a sequence of singular value decompositions. As a result, this approach requires a significant amount of memory. The memory of our existing computational resources limits us to a maximum number of around 30,000 frames for `TT-SVD`.

One way to reduce the memory and computational requirements of `TT-SVD` is adopting an incremental approach. `TT-ICE`* (Aksoy et al., 2022, Alg. 3.2) is an incremental algorithm that computes the TT-decomposition for a stream of tensors with a given target accuracy. To train the tensor network, we create batches of 2500 frames from gameplay sequences and update the basis vectors stored in TT-cores using `TT-ICE`* . Although `TT-ICE`* does not require a fixed batch size as shown in Aksoy et al. (2022), we train the tensor network using a fixed batch size to mimic the training of the NN based methods. Once a pass is completed over a batch of images, their latent representations are stacked to $\mathcal{G}_6$. Also note that `TT-ICE`* explores and increases the number of the basis vectors for each dimension of the tensor stream adaptively based on the user-specified error tolerance. This removes the need to manually tune the size of the latent space L for the encoded images. Next, we discuss how out-of-sample gameplay frames are mapped to the latent space by each encoding method.

3.2 Mapping unseen data to latent space

For AE/VAE, the latent space representation of unseen data is acquired directly through a forward pass of the trained encoder with the desired image. On the other hand for TT, the notion of forward pass is replaced with sequential projection onto cores.

Note that the first 5 TT-cores contain reshaped orthonormal vectors when TT-cores are trained with `TT-ICE`* . We obtain r_i orthonormal vectors for each dimension by reshaping the first 5 TT-cores as

$$U_i = \texttt{reshape}\left(\mathcal{G}_i, [r_{i-1}n_i, r_i]\right). \tag{3}$$

These orthonormal vectors are then used to encode any new images from the same game type. We obtain the latent embedding of images by simply projecting the appropriately reshaped images onto U_i. The pseudocode of this procedure is provided in Algorithm 1[1].

The AE, VAE, and TT all allow for reconstructing frames from the latent space. As such, we can visualize the reconstructed frames to get a sense of what is lost in encoding the frames. Figure 2 shows an example of reconstructed frames from the latent space for each encoding method.

Algorithm 1 Mapping unseen data to latent space by orthogonal projections onto TT-cores

Input
  $\{\mathcal{G}_i\}_{i=1}^{5}$ TT-cores trained with `TT-ICE`*
  $\mathcal{Y}_j \in \mathbb{R}^{n_1 \times \cdots \times n_5 \times N_j}$ j-th batch of frames that contain N_j frames to be projected
Output
  $g_j \in \mathbb{R}^{r_5 \times N_j}$ encodings of the j-th batch of frames
$g_j \leftarrow \texttt{reshape}\left(\mathcal{Y}_j, [n_1, n_2 \ldots n_5 N_j]\right)$
for $i = 1$ to 4 **do**
  $U \leftarrow \texttt{reshape}\left(\mathcal{G}_i, [r_{i-1}n_i, r_i]\right)$ ▷ Note that for $i = 1$ we have $r_0 = 1$
  $g_j \leftarrow U^T g_j$
  $g_j \leftarrow \texttt{reshape}\left(g_j, [r_i n_{i+1}, n_{i+2} \ldots N_j]\right)$
end for
$U \leftarrow \texttt{reshape}\left(\mathcal{G}_5, [r_4 n_5, r_5]\right)$
$g_j \leftarrow U^T g_j$

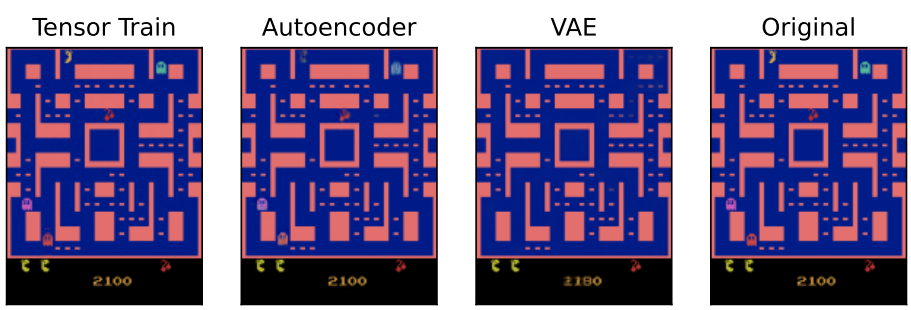


Figure 2: *Example reconstruction of a Ms.Pac-Man frame using TT (with $\varepsilon = 0.01$), AE and VAE. TT reconstruction is visually the most similar to the original frame. The size of the latent space for all three models is 11583, which is* 11.5% *of the original size* ($210 \times 160 \times 3$).

3.3 Action predictor

To extract the game-specific playing strategy, we employ a simple MLP based architecture. We stack the latent representation of the last 4 frames together and obtain a $4L$ vector as input to the action predictor.

[1]This algorithm is adapted from Aksoy et al. (2022)

Then the action predictor outputs the predicted action for a given timestep. Using the previous four frames as input allows the model to use some amount of temporal information and is a common choice for experiments conducted on Atari games (see, e.g., Mnih et al. (2013)).

The action predictor consists of 5 hidden layers with 50 neurons each. Every hidden layer is then followed by a batch norm and ReLU operation. Finally, the output layer has size equal to the number of possible actions for each game, which is interpreted as the probability of taking each action. Next, we present the computational experiments we have conducted.

4 Experiments

In this section, we describe our comparative study, the dataset we selected for the experiments, evaluation metrics we used, and results we obtained.

4.1 Dataset

The dataset consists of game frames from demonstrations from a set of Atari games, in particular: BeamRider, Breakout, MsPacman, Pong, Qbert, Seaquest, and SpaceInvaders. Our data is collected using trained Deep Q-Networks (Mnih et al., 2015), which are obtained from the RL-Baselines Zoo package (Raffin, 2018) and further trained using the Stable Baselines platform (Hill et al., 2018) for 10^7 samples (states and actions), using the tuned hyperparameters from RL-Baselines Zoo. Rather than use downsized grayscale $84 \times 84 \times 1$ images (before stacking) as is often done in the literature, we instead use the original full-sized RGB images ($210 \times 160 \times 3$), since our work deals directly with image compression and dimensionality reduction. For each game, we generate 200 trajectories, a subset of which is used as a training set and a subset of which is used as a validation set. More details about the split are given in Section 4.2. Another 50 trajectories are generated as a test set, which is used to evaluate the action prediction accuracy of the final trained models. We define a trajectory as a sequence of states (frames), rewards, and actions taken by the RL agent, starting at the beginning of the game and ending when the RL agent loses a life in the game or the end of the game is reached. The dataset can be found at Chen (2023).

4.2 Setup

The procedure for our experiments is as follows:

1. Train an AE, VAE, and TT on the frames from a set of gameplay videos. The dimensionality of the latent space L for the TT is determined automatically by `TT-ICE`* . The dimensionality of the latent space for AE and VAE is then fixed to be equal to L in order to provide a fair comparison of the methods.

   Furthermore, the VAE and AE are trained using a range of L2 regularization coefficients and initial learning rates.

2. For the AE and VAE, an additional portion of the gameplay videos are held out for validation. The validation reconstruction errors are used to select a VAE and AE for encoding. Note that for TT-decomposition no such validation procedure is needed.

3. Using the encodings generated by the TT or the AE / VAE as input, we train simple feed-forward networks to predict the actions taken by the trained reinforcement learning agent from which data is generated, using a range of hyperparameters.

4. For each of the AE, VAE, and TT encodings, we select the predictor that yields the lowest validation action prediction loss, as measured by cross-entropy loss on a separate validation set.

In our work, we consider training the AE, VAE, and TT separately to first learn an encoding, before training a predictor on the encodings directly. This is in contrast to end-to-end approaches, where the encodings would be trained jointly with the predictors. An advantage of training the encoders separately from the action

	Limited Data	Moderate Data
Training Set for Training Encoder and Predictor	5	160
Validation Set for Selecting Best Encoder (ONLY AE/VAE)	1	20
Validation Set for Selecting Best Predictor	1	20
Test Set for Evaluating Final Models	50	50

Table 1: *Summary of the number of trajectories used to train the models in the limited and moderate data cases. Note that each row corresponds to separate trajectories.*

predictors is reduced training time, as the expensive encoders are not further finetuned for the particular task of behavioral cloning. A second advantage is that the encodings are smaller than the full frames, leading to lower memory and storage requirements. This approach mimics the approach taken in several reinforcement learning papers, which have found that decoupling training an encoder from training the reinforcement learning agent is an effective approach - examples include Pari et al. (2021); Parisi et al. (2022); Stooke et al. (2021); Yuan et al. (2022); Shah & Kumar (2021).

The predictors and AE / VAE were set up and trained using PyTorch and trained on NVIDIA Tesla V100 GPUs. The TT is trained either using GPU (TT-GPU, NVIDIA Tesla V100) or CPU (TT-CPU, Intel Xeon Gold 6154 processors), using the TT-ICE* package - timings are reported for both. Note that the only difference between TT-GPU and TT-CPU is the device that the computation took place. In parallel to the findings in Aksoy et al. (2022), we have empirically found that this approach is most effective when $n_i = \mathcal{O}(10)$ for $i = 1, \ldots, 5$ and therefore reshaped the image batches to $15 \times 14 \times 16 \times 10 \times 3 \times N$.

In this work, we consider two cases: 1) a limited data regime where only 5 gameplay trajectories are used as training data for each game and 2) a moderate data regime where 160 gameplay trajectories are used as training data for each game. A summary of the number of trajectories used in the limited and moderate data settings is shown in Table 1. In addition to these two scenarios, we also provide a more detailed investigation on the effects of changing the latent space size (Appendix A.4) and the amount of training data (Appendix A.5) on action prediction accuracy and computational speedup.

Action predictors are trained with the Adam-W optimizer with a cosine learning rate decay for fifty epochs in the low data case and five epochs in the moderate data case. These predictors are trained using the same range of initial learning rates and weight decay penalties as the ones used to train the AE and VAE encoders (shown in Appendix A.1). An additional 1 or 20 trajectories (in the limited and moderate data regimes, respectively) not used during training are then used as a validation set to evaluate the trained predictors, with the predictors yielding the lowest cross-entropy action prediction loss selected.

We also include the results of a predictor trained using the original, full-sized RGB frames, of dimension $210 \times 160 \times 3$. Because the "no encoding" predictor uses the full-sized images, it is expected that these accuracies will be higher than the predictors using the encodings - these are included as a comparison as an upper bound on accuracy we can expect.

Additional details on training and the neural network architectures can be found in the Appendix A.1.

4.3 Metrics

We compare performance of the methods using three metrics: training time, action prediction accuracy, and normalized gameplay score. We then report the averaged results for each of those metrics for each regime in Section 4.4.

Training Time. We compare the time required to train each framework against the time needed to train a predictor on full frames.

Reconstruction Quality. We measure the reconstruction quality of the TT against an AE and VAE using peak-signal-to-noise ratio (PSNR) and mean-squared error (MSE) as metrics.

Action prediction accuracy. We compare the accuracy of the actions suggested by each framework to actions taken by the RL agent.

Normalized gameplay score. We normalize gameplay scores achieved by each framework so that a score of 0 corresponds to performance of an agent taking random actions and a score of 1 corresponds to performance of the RL agent. In particular, given the score s achieved by a behavioral cloned agent, the mean score s_{RL} achieved by the RL agent and the mean score s_{random} achieved by an agent taking random actions, we calculate the normalized score to be

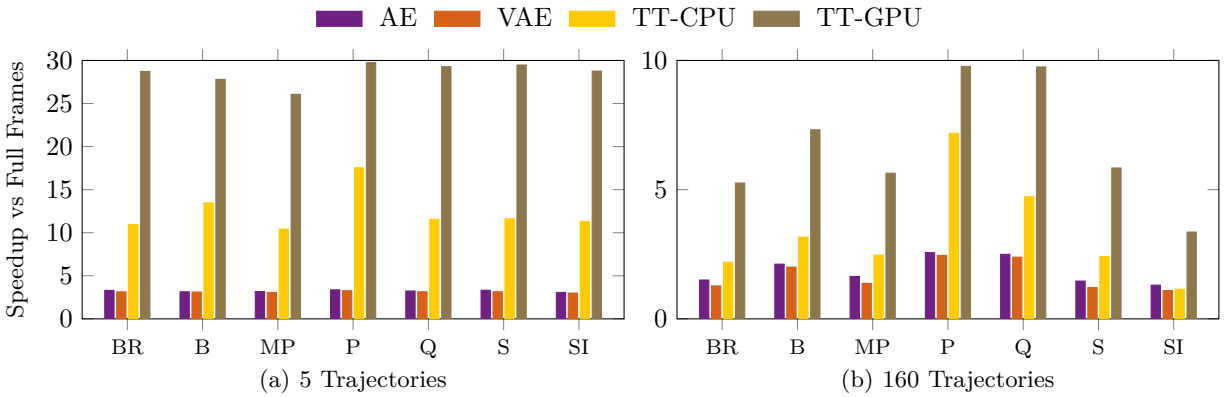
$$s_{\text{normalized}} = \frac{s - s_{\text{random}}}{s_{\text{RL}} - s_{\text{random}}}.$$

The training time for each framework can further be broken down into the time it takes to train the encoder, the time it takes to encode the data, and the time it takes to train the action predictor given the encodings. These results are shown in Appendices A.2 and A.3.

4.4 Results

4.4.1 Limited Data Regime

The benefits of bypassing the need for training a neural network encoder can be observed in the total training time of the different methods, as shown in Figure 3. In particular, the training time of the TT-based predictor trained on GPU (TT-GPU) is 8.2× to 9.6× faster compared to the AE and VAE-based predictors and 26.1× to 29.7× faster compared to the predictor trained with no encoding. When the TT encoding is trained on CPU (but with the predictor still trained on GPU), the TT-predictor is still 3.3× to 5.4× faster than the AE and VAE predictors trained on GPU and 10.4× to 17.5× faster than the predictor trained with no encoding.



(a) 5 Trajectories (b) 160 Trajectories

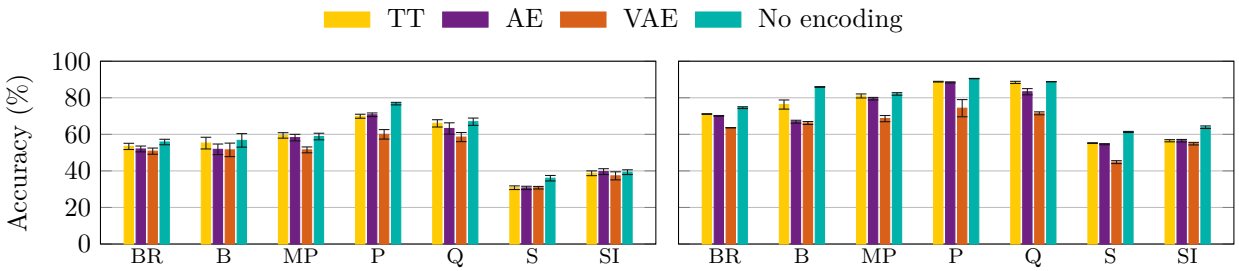
Figure 3: *Speedup of the end-to-end training procedure of the AE, VAE, and TT-based predictors compared to a predictor trained using the full RGB frames. Extracting a latent space reduces end-to-end training time in all cases. In the low data regime, using the TT-GPU leads to a* 26.1× *to* 29.7× *faster training time than using the full RGB frames and a* 8.2× *to* 9.6× *speedup compared to the AE and VAE. In the moderate data case (*160 *trajectories), using TT-GPU leads to a* 3.4× *to* 9.8× *faster training time compared to using the full RGB frames and a* 2.6× *to* 4.8× *speedup compared to the AE and VAE. Note that the difference in speedups between the* 5 *trajectory case and* 160 *trajectory case is due to the difference in training epochs used in each case (*50 *in the* 5 *trajectory case and* 5 *in the* 160 *trajectory case); the speedups are similar when the number of training epochs is held fixed while the number of training trajectories is changed, as shown in Appendix A.5. BR:Beam Rider, B:Breakout, MP:Ms.Pacman, P:Pong, Q:Qbert, S:Seaquest, SI:Space Invaders.*

.

This speedup comes with an improvement in reconstruction quality, as shown in Table 2; the TT leads to a lower MSE and higher PSNR compared to the AE and VAE in six of the seven games. This speedup also comes at no cost in terms of behavioral cloning accuracy (Figure 4a) and achieved gameplay scores (Figure 5a), compared to training using encodings generated by an AE or VAE.

	MSE			PSNR		
	TT	AE	VAE	TT	AE	VAE
BeamRider (BR)	**6.3**	30.5	47.6	**46.8**	35.3	33.0
Breakout (B)	**33.0**	67.6	80.9	**39.4**	32.9	30.3
MsPacman (MP)	**27.1**	58.0	103.3	**38.7**	32.5	28.9
Pong (P)	2.0	**0.7**	56.9	46.7	**61.6**	36.9
Qbert (Q)	**6.8**	33.5	68.5	**45.6**	34.3	30.8
Seaquest (S)	**13.9**	30.8	74.0	**38.6**	33.9	29.8
SpaceInvaders (SI)	**80.3**	144.5	176.4	**32.4**	28.4	26.2

Table 2: *Reconstruction quality of the TT, AE, and VAE in the low data regime. The reported statistics are MSE (using 255 as maximum pixel value) and PSNR of the reconstructions versus the original frames. For each game, the bolded values are the smallest MSE and highest PSNR.*



(a) 5 Trajectories (b) 160 Trajectories

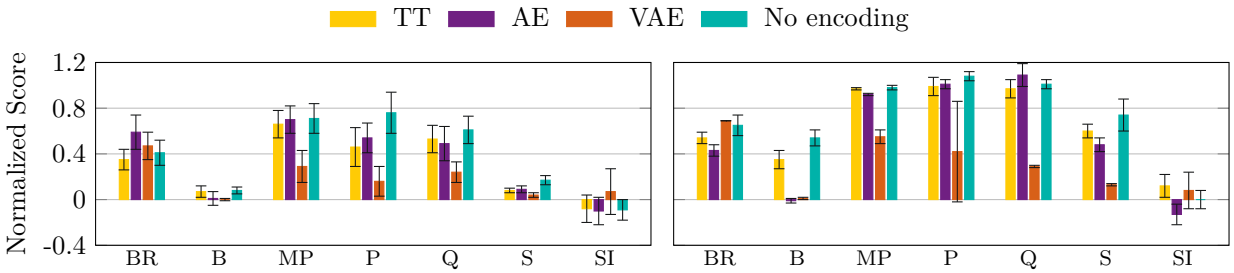
Figure 4: *Action prediction accuracies of trained predictors using encodings from the TT, AE, and VAE, as well as a trained predictor using the full-sized original RGB frames. Training a predictor using TT encodings yields similar and sometimes better prediction accuracies as training a predictor using AE or VAE encodings. Error bounds are calculated as two times the standard error of gameplay scores calculated across ten (in the* 5 *trajectory case) or three (in the* 160 *trajectory case) experiments.*

4.4.2 Moderate Data Regime

In the case where 160 trajectories are used for training data, we find once again that the TT-based predictor provides a large speedup relative to the AE, VAE, and full frame predictors. In particular, as shown in Figure 3, using the TT-encoder trained on GPU led to a 3.4× to 9.8× faster training time compared to the no-encoding predictor. Compared to the AE and VAE, the end-to-end training time of the TT-based predictor is 2.6× to 4.8× faster. Training the TT on CPU still usually leads to a speedup over the AE and VAE on the GPU (0.9× to 2.9× faster), showing the computational efficiency of TT-ICE even on CPU compared to GPU-based deep learning.

These results come with an improvement in reconstruction quality and no cost in terms of behavioral cloning accuracy or gameplay scores. The reconstructions generated by the TT have lower error compared to the reconstructions generated by the AE and VAE, as shown in Table 3. Furthermore, we find improved behavioral cloning accuracy of the predictor trained using the TT encodings versus the predictors trained on the AE or VAE encodings for some of the games, as seen in Figure 4b. The gameplay scores achieved by each of the encoding methods is comparable, as shown in Figure 5b.

Note that in some games, the gameplay scores achieved by the predictors, including the one trained with no encoding, are much lower than the scores achieved by the RL agent. This is likely due to an issue encountered in imitation learning where the distribution of data used to train the agent differs from the distribution of data encountered while testing the agent, since the agent's actions directly affect the future states it encounters. This leads to compounding errors and hurts the performance of the imitation learning agent (Ross & Bagnell, 2010). These low scores are also consistent with Kanervisto et al. (2020), which



(a) 5 Trajectories (b) 160 Trajectories

Figure 5: *Normalized gameplay scores achieved by each of the behavioral cloning models, trained using either encodings from the TT, AE, VAE, or the original RGB images. A score of* 0 *corresponds to performance of an agent taking random actions and a score of* 1 *corresponds to performance of the RL agent. Training a predictor using TT encodings yields similar gameplay scores as training a predictor using AE or VAE encodings. Error bounds are calculated as two times the standard error of gameplay scores calculated across ten (in the* 5 *trajectory case) or three (in the* 160 *trajectory case) experiments. Please refer to the caption of Figure 4 for the game abbreviations.*

	MSE			PSNR		
	TT	AE	VAE	TT	AE	VAE
BeamRider (BR)	**0.4**	18.0	26.0	**58.1**	37.6	36.4
Breakout (B)	**1.2**	7.8	19.8	**51.0**	40.1	36.8
MsPacman (MP)	**1.7**	16.2	75.7	**48.1**	37.6	30.2
Pong (P)	1.8	**0.0**	6.9	47.0	**71.4**	40.7
Qbert (Q)	**0.8**	12.1	36.4	**52.7**	39.0	33.8
Seaquest (S)	**1.0**	8.4	52.9	**49.5**	39.9	31.4
SpaceInvaders (SI)	**0.4**	18.5	80.2	**56.1**	36.5	30.1

Table 3: *Reconstruction quality of the TT, AE, and VAE in the moderate data regime. The reported statistics are the MSE (using 255 as maximum pixel value) and PSNR of the reconstructions versus the original frames. For each game, the bolded values are the smallest MSE and highest PSNR.*

found that behavioral cloning from human gameplay was unable to match human-level gameplay scores for the Atari games considered in that paper.

4.4.3 Additional Results

Appendix A.2 demonstrates a more detailed investigation into the timing differences between the methods. The speedups achieved by the TT compared to the AE and VAE are split into three components:

1. Time to train the encoder.
2. Time to encode training and validation data used to train the action predictor.
3. Time to train the action predictor.

These results show that the most time-intensive task is training the encoder. As such, reducing this training time using a TT instead of a AE or VAE leads to a large reduction in overall training time. In Appendix A.3, we look at the encoding time of the various models, since a faster encoding time leads to a faster inference time. We find that the encoding time for TT-GPU is between 0.1× and 8.9× that of the AE and VAE in the 5 trajectory case and between 1.5× and 8.9× that of the AE and VAE in the 160 trajectory case. The time it takes to encode data is a drawback of using the TT, which is an area for future research. However, even with the long encoding time, the overall time for training the TT on GPU is still much lower than that of the AE and VAE since training the encoding is the most time-intensive task.

To ensure that our results are consistent across different settings, we also examine how changing the latent space size (Appendix A.4) and the number of training trajectories (Appendix A.5) affects the training time and action prediction accuracy of the AE, VAE, and TT. The results are consistent: we find that as the latent space size and number of training trajectories are changed, TT achieves similar prediction accuracies as the AE and VAE but provides a speedup in training time.

5 Conclusion

In this work, we have explored using tensor networks as a method for learning a low-dimensional representation of image data for behavioral cloning. In particular, we compare the computational speed of using TT against AEs and VAEs for extracting this low dimensional representation, finding a 2.6× to 9.6× speedup compared to AEs or VAEs. Furthermore, the TT encoding can be efficiently trained on CPU as well, which leads to comparable or better training times than the AE and VAE trained on GPU (0.9× to 5.4× faster). This speedup comes at no cost in terms of behavioral cloning accuracy or the gameplay scores achieved by the behavioral cloned agents. As such, the TT-based approach enables learning while having limited access to GPUs, as it allows for the expensive dimensionality reduction step to be performed on CPUs at a comparable or faster time than using an AE or VAE on GPUs. Given the large computational requirements of learning directly from video data, future work will aim to extend this methodology to tackling larger game environments for which the large input space is a computational bottleneck.

References

Doruk Aksoy, David J Gorsich, Shravan Veerapaneni, and Alex A Gorodetsky. An incremental tensor train decomposition algorithm. *arXiv preprint arXiv:2211.12487*, 2022.

Alexander Amini, Wilko Schwarting, Guy Rosman, Brandon Araki, Sertac Karaman, and Daniela Rus. Variational autoencoder for end-to-end control of autonomous driving with novelty detection and training de-biasing. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 568–575. IEEE, 2018.

Bowen Baker, Ilge Akkaya, Peter Zhokov, Joost Huizinga, Jie Tang, Adrien Ecoffet, Brandon Houghton, Raul Sampedro, and Jeff Clune. Video pretraining (vpt): Learning to act by watching unlabeled online videos. *Advances in Neural Information Processing Systems*, 35:24639–24654, 2022.

Eli Bronstein, Mark Palatucci, Dominik Notz, Brandyn White, Alex Kuefler, Yiren Lu, Supratik Paul, Payam Nikdel, Paul Mougin, Hongge Chen, et al. Hierarchical model-based imitation learning for planning in autonomous driving. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 8652–8659. IEEE, 2022.

Daniel Brown, Russell Coleman, Ravi Srinivasan, and Scott Niekum. Safe imitation learning via fast bayesian reward inference from preferences. In *International Conference on Machine Learning*, pp. 1165–1177. PMLR, 2020.

Xiaoyu Cai, Qiongli Jing, Bo Peng, Yuanyuan Zhang, Yuting Wang, Ju Tang, et al. Automatic traffic state recognition based on video features extracted by an autoencoder. *Mathematical Problems in Engineering*, 2022, 2022.

Brian Chen. Atari Games Dataset. https://deepblue.lib.umich.edu/data/concern/data_sets/gq67jr854, 2023.

Brian Chen, Siddhant Tandon, David Gorsich, Alex Gorodetsky, and Shravan Veerapaneni. Behavioral cloning in atari games using a combined variational autoencoder and predictor model. In *2021 IEEE Congress on Evolutionary Computation (CEC)*, pp. 2077–2084. IEEE, 2021a.

Xin Chen, Sam Toyer, Cody Wild, Scott Emmons, Ian Fischer, Kuang-Huei Lee, Neel Alex, Steven H Wang, Ping Luo, Stuart Russell, et al. An empirical investigation of representation learning for imitation. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021b.

Krzysztof Fonał and Rafał Zdunek. Distributed and randomized tensor train decomposition for feature extraction. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. IEEE, 2019.

Timur Garipov, Dmitry Podoprikhin, Alexander Novikov, and Dmitry Vetrov. Ultimate tensorization: compressing convolutional and FC layers alike. *arXiv preprint arXiv:1611.03214*, 2016.

Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Stable baselines. https://github.com/hill-a/stable-baselines, 2018.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, 50(2):1–35, 2017.

Miguel Jaques, Michael Burke, and Timothy M Hospedales. Newtonianvae: Proportional control and goal identification from pixels via physical latent spaces. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4454–4463, 2021.

Anssi Kanervisto, Joonas Pussinen, and Ville Hautamäki. Benchmarking end-to-end behavioural cloning on video games. In *2020 IEEE conference on games (CoG)*, pp. 558–565. IEEE, 2020.

Heecheol Kim, Yoshiyuki Ohmura, and Yasuo Kuniyoshi. Transformer-based deep imitation learning for dual-arm robot manipulation. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 8965–8972. IEEE, 2021.

Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1511.06530.

Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014. URL http://arxiv.org/abs/1312.6114.

Jean Kossaifi, Aran Khanna, Zachary Lipton, Tommaso Furlanello, and Anima Anandkumar. Tensor contraction layers for parsimonious deep nets. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 26–32, 2017.

Jean Kossaifi, Zachary C Lipton, Arinbjorn Kolbeinsson, Aran Khanna, Tommaso Furlanello, and Anima Anandkumar. Tensor regression networks. *Journal of Machine Learning Research*, 21(123):1–21, 2020.

V Lebedev, Y Ganin, M Rakhuba, I Oseledets, and V Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. In *3rd International Conference on Learning Representations, ICLR 2015-Conference Track Proceedings*, 2015.

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2016.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2018.

Angshul Majumdar. Blind denoising autoencoder. *IEEE transactions on neural networks and learning systems*, 30(1):312–317, 2018.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 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*, 518:529–533, 2015.

Alexander Novikov, Dmitrii Podoprikhin, Anton Osokin, and Dmitry P Vetrov. Tensorizing neural networks. *Advances in neural information processing systems*, 28, 2015.

OpenAI, Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning. 2019. URL https://arxiv.org/abs/1912.06680.

Ivan V Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011.

Jyothish Pari, Nur Muhammad, Sridhar Pandian Arunachalam, Lerrel Pinto, et al. The surprising effectiveness of representation learning for visual imitation. *Robotics: Science and Systems XVIII*, 2021.

Simone Parisi, Aravind Rajeswaran, Senthil Purushwalkam, and Abhinav Gupta. The unsurprising effectiveness of pre-trained vision models for control. In *International Conference on Machine Learning*, pp. 17359–17371. PMLR, 2022.

Emmanuel Pintelas, Ioannis E Livieris, and Panagiotis E Pintelas. A convolutional autoencoder topology for classification in high-dimensional noisy image datasets. *Sensors*, 21(22):7731, 2021.

Marsela Polic, Ivona Krajacic, Nathan Lepora, and Matko Orsag. Convolutional autoencoder for feature extraction in tactile sensing. *IEEE Robotics and Automation Letters*, 4(4):3671–3678, 2019.

Antonin Raffin. Rl baselines zoo. https://github.com/araffin/rl-baselines-zoo, 2018.

Stéphane Ross and Drew Bagnell. Efficient reductions for imitation learning. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 661–668. JMLR Workshop and Conference Proceedings, 2010.

Pierre Sermanet, Corey Lynch, Jasmine Hsu, and Sergey Levine. Time-contrastive networks: Self-supervised learning from multi-view observation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 486–487. IEEE, 2017.

Rutav M Shah and Vikash Kumar. Rrl: Resnet as representation for reinforcement learning. In *International Conference on Machine Learning*, pp. 9465–9476. PMLR, 2021.

Manish Sharma, Panos P Markopoulos, Eli Saber, M Salman Asif, and Ashley Prater-Bennette. Convolutional auto-encoder with tensor-train factorization. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 198–206, 2021.

Hao Shen, Weikang Wan, and He Wang. Learning category-level generalizable object manipulation policy via generative adversarial self-imitation learning from demonstrations. *IEEE Robotics and Automation Letters*, 7(4):11166–11173, 2022.

K Sreelakshmi and Renjith V Ravi. An encryption-then-compression scheme using autoencoder based image compression for color images. In *2020 7th International Conference on Smart Structures and Systems (ICSSS)*, pp. 1–5. IEEE, 2020.

Adam Stooke, Kimin Lee, Pieter Abbeel, and Michael Laskin. Decoupling representation learning from reinforcement learning. In *International Conference on Machine Learning*, pp. 9870–9879. PMLR, 2021.

Mojtaba Taherisadr, Mohsen Joneidi, and Nazanin Rahnavard. Eeg signal dimensionality reduction and classification using tensor decomposition and deep convolutional neural networks. In *2019 IEEE 29th International Workshop on Machine Learning for Signal Processing (MLSP)*, pp. 1–6. IEEE, 2019.

Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

Tianyu Wang, Nikhil Karnwal, and Nikolay Atanasov. Latent policies for adversarial imitation learning. *arXiv preprint arXiv:2206.11299*, 2022.

Yasi Wang, Hongxun Yao, and Sicheng Zhao. Auto-encoder based dimensionality reduction. *Neurocomputing*, 184:232–242, 2016.

Xuewen Yang, Yingru Liu, and Xin Wang. Reformer: The relational transformer for image captioning. In *Proceedings of the 30th ACM International Conference on Multimedia*, pp. 5398–5406, 2022.

Jianbo Yu and Xingkang Zhou. One-dimensional residual convolutional autoencoder based feature learning for gearbox fault diagnosis. *IEEE Transactions on Industrial Informatics*, 16(10):6347–6358, 2020.

Rose Yu, Stephan Zheng, Anima Anandkumar, and Yisong Yue. Long-term forecasting using higher order tensor rnns. *arXiv preprint arXiv:1711.00073*, 2017a.

Xiyu Yu, Tongliang Liu, Xinchao Wang, and Dacheng Tao. On compressing deep models by low rank and sparse decomposition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7370–7379, 2017b.

Zhecheng Yuan, Zhengrong Xue, Bo Yuan, Xueqian Wang, Yi Wu, Yang Gao, and Huazhe Xu. Pre-trained image encoder for generalizable visual reinforcement learning. *Advances in Neural Information Processing Systems*, 35:13022–13037, 2022.

Jaime Zabalza, Jinchang Ren, Jiangbin Zheng, Huimin Zhao, Chunmei Qing, Zhijing Yang, Peijun Du, and Stephen Marshall. Novel segmented stacked autoencoder for effective dimensionality reduction and feature extraction in hyperspectral imaging. *Neurocomputing*, 185:1–10, 2016.

Wenjie Zhang, Jiqing Han, and Shiwen Deng. Heart sound classification based on scaled spectrogram and tensor decomposition. *Expert Systems with Applications*, 84:220–231, 2017.

Bin Zhao, Xuelong Li, and Xiaoqiang Lu. Tth-rnn: Tensor-train hierarchical recurrent neural network for video summarization. *IEEE Transactions on Industrial Electronics*, 68(4):3629–3637, 2020.

Appendix

A Additional Details on Training

A.1 Feature Extraction Step

Architectural details for the encoding part of our AEs and VAEs are laid out in Table 4. The decoding portion of these networks mirrors the encoding portion, with convolutional layers replaced by transposed convolutional layers.

The AEs and VAEs are hyperparameter tuned individually for each game over initial learning rates of $10^{-2}, 10^{-3}$, and 10^{-4} and L2 weight decay penalties of $1, 10^{-2}$, and 10^{-4}, after which the best model is selected based on validation error and used for encoding. The networks are trained using the Adam-W optimizer (Loshchilov & Hutter, 2018) with cosine learning rate decay without restarts (Loshchilov & Hutter, 2016) for fifty epochs in the low data case and five epochs in the moderate data case. The AE is trained to minimize reconstruction error measured via mean squared error while the VAE is trained using the standard

Table 4: *Architecture for the encoding part of AEs and VAEs. After each intermediate layer we also apply batch norm followed by LeakyReLU - these are excluded from the table for simplicity. The asymmetric padding in the first layer is to account for the shape of the input.*

Layer #	Layer Type	Inp. Ch.	Out. Ch.	Kernel Size	Stride	Padding
1	Conv	3	32	3	2	(4, 1)
2	Conv	32	64	3	2	(1, 1)
3	Conv	64	64	3	2	(1, 1)
4	Conv	64	16	1	1	(0, 0)
5	Flatten	-	-	-	-	-
6	Linear	-	-	-	-	-

ELBO loss (Kingma & Welling, 2014). An additional 1 or 20 trajectories (in the limited and moderate data regimes, respectively) are used as a validation set in order to hyperparameter tune the networks. Note that the TT does not require this step of validation and does not use this additional data.

A.2 Breakdown of Speedups

In Figures 6 and 7, we further break down the timings for training the AE, VAE, and TT-based predictors as follows:

- ***Training the encoder:*** For the AE and VAE, this time is the total time required to perform a hyperparameter optimization by training nine AEs / VAEs with different network hyperparameters and selecting one based on the validation loss measured on a separate validation set. Recall that TT encoder does not require such a procedure. For the TT, this is the time required to perform the TT decomposition (using `TT-ICE`*) to extract the latent space representation of the input data.
- ***Encoding data:*** Time to pass all the training and validation data through the AE, VAE, and TT in order to extract latent space representations.
- ***Training the action predictor:*** Time to train the action predictor on the encodings extracted in the previous step. The architecture of the predictor is same across all three methods. For each model and for each game, nine predictors are trained using different initial learning rates (10^{-2}, 10^{-3}, and 10^{-4}) and L2 regularization penalties (1, 10^{-2}, and 10^{-4}). The predictor yielding the lowest cross-entropy action prediction loss measured on another separate validation set is selected.

Note that the most time-intensive task is training the encoding. As such, reducing the time it takes to train the encoding by using a TT instead of an AE or VAE leads to a significant reduction in overall training time.

A.3 Encoding Times

Encoding times of the various models are compared in Table 5. In the 5 trajectory case, the encoding time for TT-GPU is between 0.1× and 8.9×. In the 160 trajectory case, the encoding time for TT-GPU is between 1.5× and 8.9× that of the AE and VAE. An area of future work will be to reduce the encoding times required for the TT, since this affects the inference times of these models.

A.4 Effect of Changing Epsilon

We conduct experiments testing the effect of changing the size of the latent space for the TT, AE, and VAE. In our experiments, the latent size of the AE and VAE is set up to match the latent size of the TT. In Table 6, the latent size as epsilon is changed is displayed.

In Figure 8, the effect of changing epsilon / the latent size on prediction accuracy is displayed. The general pattern appears to be that decreasing epsilon / increasing the latent size leads to a higher prediction accuracy,

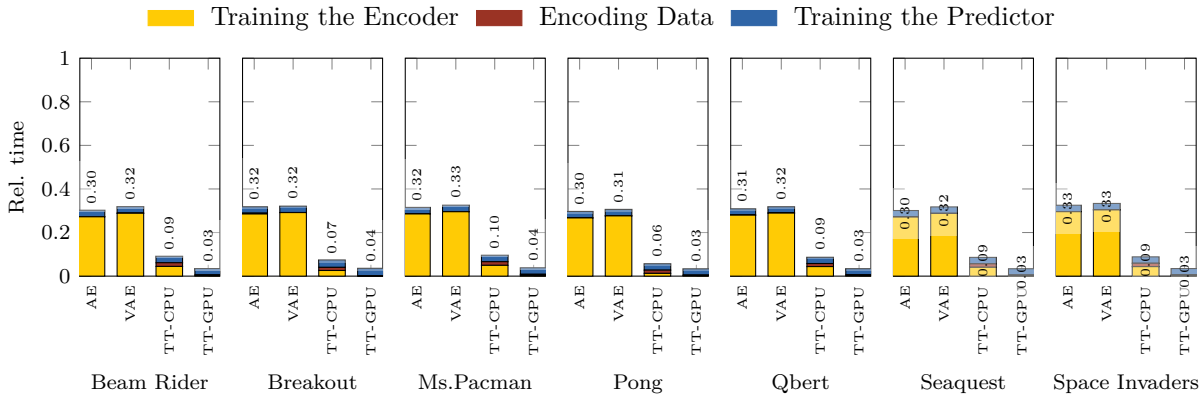


Figure 6: *Time for end-to-end training procedure in the limited data regime, averaged across all experiments, as a fraction of the time required to train the full-frame predictor. Extracting a latent space before training a predictor reduces end-to-end training time in all cases, but using the TT-GPU leads to a* 26.1× *to* 29.7× *reduction in training time. Compared to the AE / VAE, using the TT-GPU leads to a* 8.2× *to* 9.6× *speedup.*

	5 Trajectories				160 Trajectories			
	AE	VAE	TT-CPU	TT-GPU	AE	VAE	TT-CPU	TT-GPU
BeamRider	3.8	3.8	160.8	8.3	152.5	138.4	2373.0	682.2
Breakout	8.6	1.0	30.9	0.9	31.1	26.1	247.5	55.7
MsPacman	2.8	2.8	115.3	25.2	91.3	92.0	1469.5	488.2
Pong	13.0	12.4	481.6	3.1	453.9	433.9	3723.4	685.0
Qbert	1.9	1.9	67.1	7.4	60.5	60.0	641.2	122.3
Seaquest	3.3	3.2	131.6	3.0	127.1	130.7	2035.8	642.1
SpaceInvaders	1.8	1.8	66.0	7.1	76.3	71.2	1545.3	635.4

Table 5: *Time to encode the training and validation data in seconds using the various encodings. In the* 5 *trajectory case, the encoding time for TT-GPU is between* 0.1× *and* 8.9× *that of the AE and VAE. In the* 160 *trajectory case, the encoding time for TT-GPU is between* 1.5× *and* 8.9× *that of the AE and VAE.*

Epsilon	0.01	0.02	0.05	0.10	0.15
BeamRider	14645	9867	3324	297	72
Breakout	3561	957	50	18	3
MsPacman	11962	6421	867	15	2
Pong	1287	138	4	1	1
Qbert	2267	1560	308	43	10
Seaquest	15163	8170	1338	37	1
SpaceInvaders	17191	13795	8035	3994	2097

Table 6: *Latent size as epsilon for the TT is changed. The reported numbers are an average across three seeds (which use different training data).*

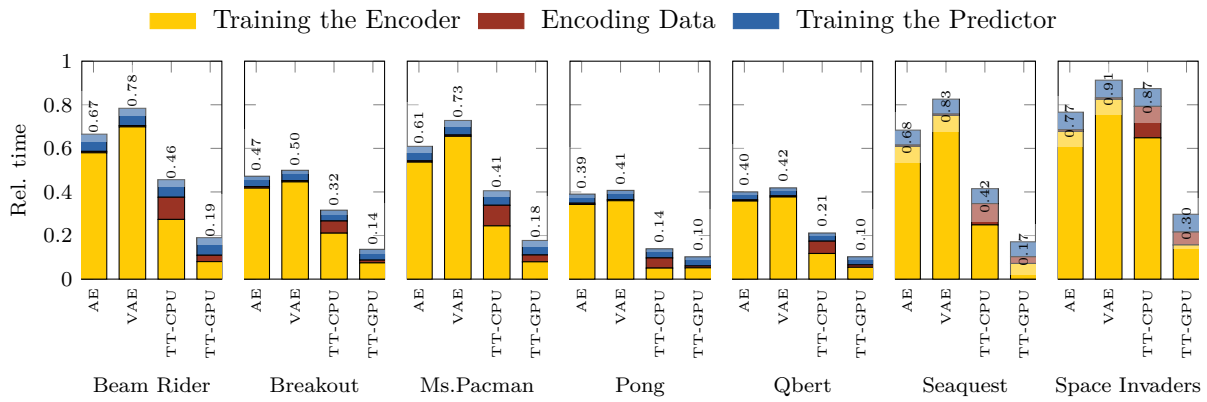


Figure 7: *Time for the end-to-end training procedure in the moderate data regime, averaged across all experiments, as a fraction of the time it takes to train a predictor using the full RGB frames. Extracting a latent space reduces end-to-end training time in all cases, but using TT-GPU leads to a* 3.4× *to* 9.8× *reduction in training time compared to no encoding and a* 2.6× *to* 4.8× *speed up relative to the AE / VAE.*

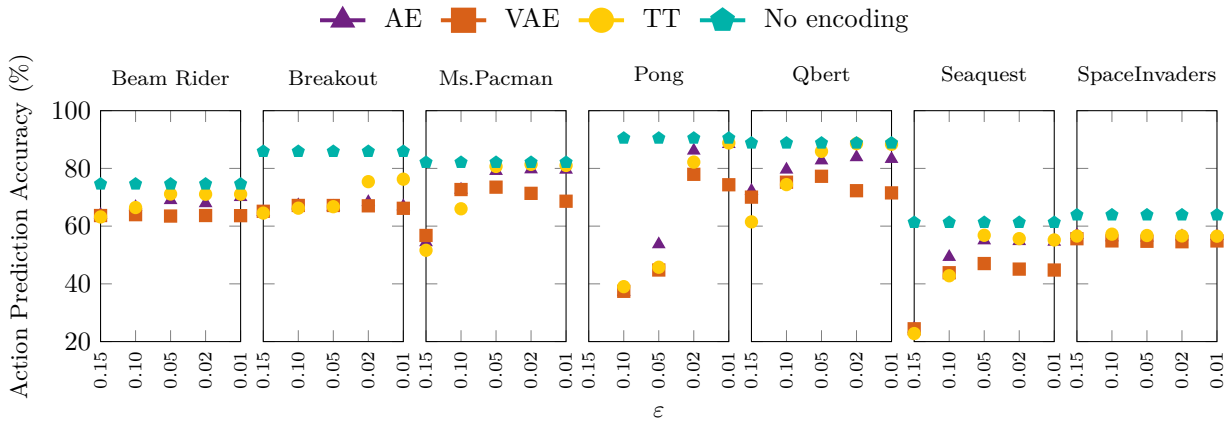


Figure 8: *Action prediction accuracy of behavioral cloning frameworks using different encoding methods at varying* ε *levels. As the latent space size is increased for the encoding-based predictors, the action prediction accuracy of the predictors tends to increase. The speedup offered by TT does not cause any performance deterioration. Furthermore, the nonlinearity offered by AE/VAE does not result in an improved latent space representation that achieves higher action prediction accuracy with a smaller latent space. Since Pong with* $\varepsilon = 0.15$ *and* $\varepsilon = 0.10$ *returns the same latent size, the accuracy for* $\varepsilon = 0.15$ *is omitted. Please refer to Table 6 for latent space sizes of* ε *levels for each different game.*

as expected. However, for some games, increasing the latent space size past a certain point leads to small or negligible improvements in prediction accuracy. The effect of changing the latent space size on timings is shown in Figure 9. Using a larger latent space size tends to increase the end-to-end training time of the models. These results are consistent with our findings in the main text - at the different epsilon levels tested, the TT-based predictor yielded similar prediction accuracies as the AE and VAE, while being faster to train.

A.5 Effect of Number of Trajectories in Training Data

We also run a small study to examine the effect of increasing the number of gameplay trajectories used to train the behavioral cloning framework on both training time and prediction accuracy. We ran this for the game of Seaquest, given the relatively low prediction accuracies we achieved compared to some of the other Atari games. As expected, Figure 10 shows that the accuracy increases as more training data is used, suggesting that better prediction accuracies can be achieved with more data.

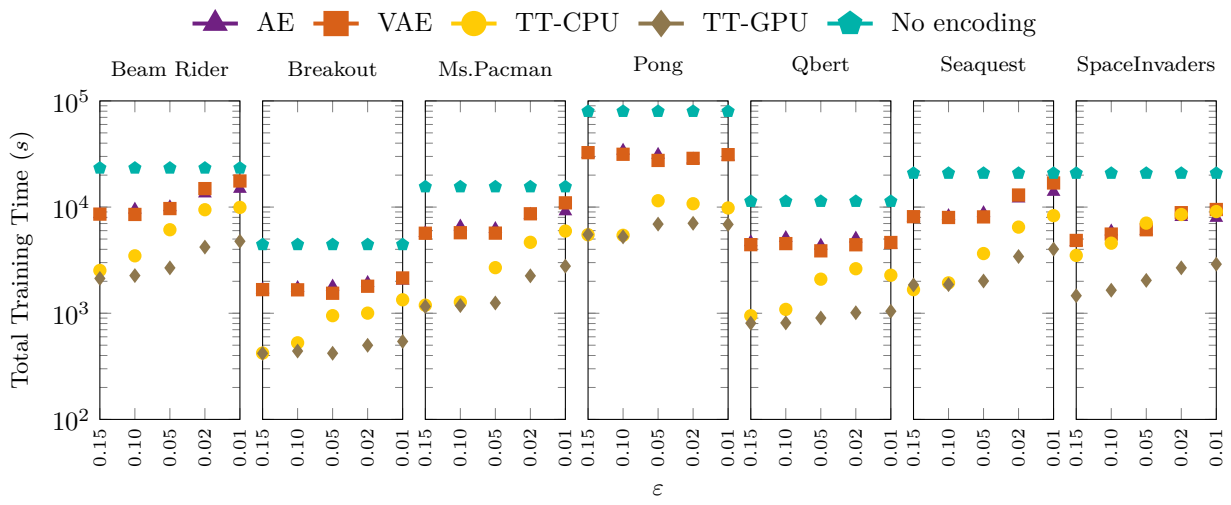


Figure 9: *Total time elapsed to train behavioral cloning frameworks using different encoding methods at varying ε levels. As the latent space size increases, the total training time for the various encoding-based predictors increases. Across all games and ε levels, TT-GPU encoding offers the fastest end-to-end training time. Please refer to Table 6 for latent space sizes of ε levels for each different game.*

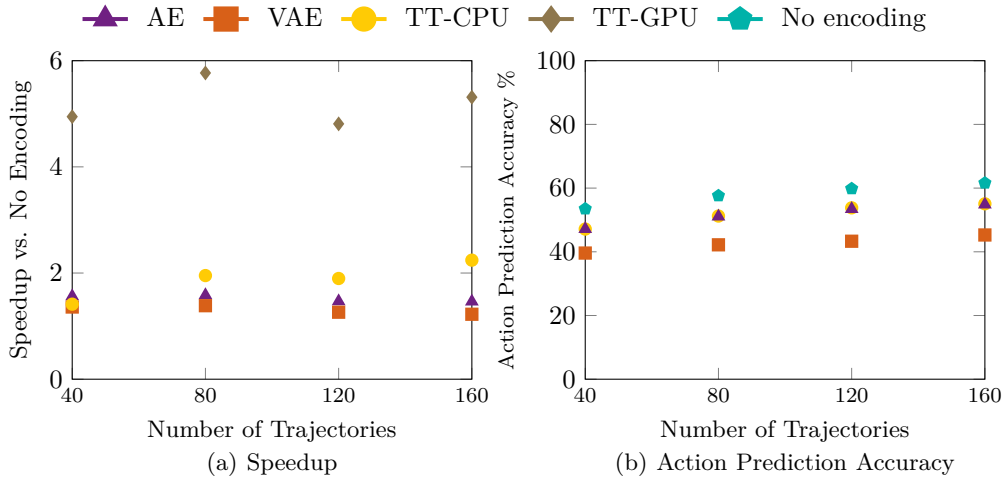


Figure 10: *Results of the experiments investigating the effect of number of trajectories used to train the behavioral cloning framework on computational speedup and action prediction accuracy using different encoding methods for the game of Seaquest. As number of training trajectories increase, action prediction accuracy gets higher. Computational benefits of* `TT-ICE`* *over the AE, VAE, or no encoding remain similar across different number of trajectories. Action prediction accuracies are only presented for TT-CPU since TT-CPU and TT-GPU result in the same latent space.*

We note that TT's speedup over the AE / VAE and no encoding remains similar as more data is used, as long as the number of training epochs is fixed. These results suggest that the timing advantage may remain for the TT even when more data is available for behavioral cloning. Note that we did not see similar speedups in Figure 3 between the 5 trajectory and 160 trajectory case, as the number of epochs used to train the AEs / VAEs was not fixed; we used 50 epochs in the 5 trajectory case and 5 epochs in the 160 trajectory case.