3D Neural Scene Representations for View-Invariant State Representation Learning

Nicholas Pfaff
nepfaff@mit.edu

Abstract

Performing tasks from visual input is a key goal in robotics. Such vision-based policies have improved significantly over the last years. Yet, they tend to be sensitive to changes in camera pose during test time. We propose to use 3D neural scene representations to learn a camera viewpoint-invariant state representation to achieve robustness to changes in camera pose. We show that there is potential in improving view-invariant state representation learning with such 3D scene representations by comparing them with multiple baselines. Noticeably, our results show that combining such 3D representations with contrastive learning is particularly effective.

A. Introduction

A key goal of robotics is to perform tasks from only visual input. Doing so requires understanding the state of the scene from images alone. Advances in computer vision enabled going from an input image to a low-dimensional latent code, describing the image [1, 4]. In robotics, it is common practice to use such latent codes as the input to a control policy [2, 7–9]. Such vision-based policies have improved significantly in recent years and can achieve robustness to many disturbances. For example, it is common for such policies to recover from scenarios where humans reset the task progress during task execution [2]. However, these policies tend to be sensitive to disturbances in the camera pose. In particular, it is common for a policy to break entirely if the camera pose is changed only slightly [2, 8]. This pose sensitivity is undesirable as it often means that new data has to be collected and the policy trained from scratch if the camera pose changes. Consequently, a policy should be insensitive to the exact camera view, enabling moving the camera slightly without significantly affecting policy performance.

Previous work attempted to address this problem by learning camera view-invariant state representations [3, 9, 10, 12]. One line of work uses contrastive learning [3] by encouraging latents corresponding to the same state to be closer to each other than latents corresponding to different states. Researchers recently attempted to use 3D neural fields [9] to learn a state representation. 3D neural fields encode the 3D scene directly rather than encoding 2D snapshots. Consequently, they encode information about 2D views not in the training data.

In this project, we investigate whether 3D neural scene representations outperform classical image encoders regarding viewpoint invariance. We consider the common robotics task planar-pushing, which aims to move an object to a goal position. Our scene is shown in Figure 1. The scene contains a red cube as the object and a green sphere as a point finger for pushing the object. This scenario is relatively tricky for classical image encoders

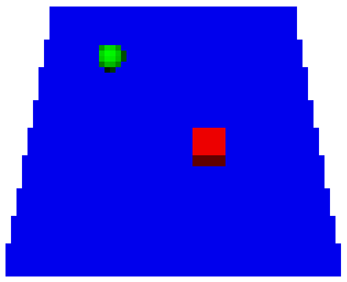
Figure 1. The planar cube environment that we use to evaluate different state encoders. The scene's state consists of the 2D finger (green ball) and block (red cube) positions. Big state changes lead to small photometric changes due to the relative size of the finger and cube compared to the background.

as most of the image contains no information about the state of the scene, which is the position of the cube and the sphere. As a result, big changes in state only lead to slight changes in the image pixels. Hence, changing the camera view can lead to bigger changes in the pixels-space than the ones caused by changes in scene state, making view invariance challenging.

We compare the state spaces learned by our 3D representations with the ones learned by classical autoencoder and contrastive learner baselines. Our results show potential in improving view-invariant state representation learning with 3D neural scene representations. Specifically, our results demonstrate that combining 3D representations with contrastive learning is particularly effective.

B. Related Work

B.1. Autoencoding

Autoencoding is a popular representation learning framework [1]. An autoencoder consists of an encoder $\mathcal{E}(\boldsymbol{I}) \mapsto \boldsymbol{s}$ that maps a signal $\boldsymbol{I}$ into a lower dimensional signal $\mathcal{L}$, where $\boldsymbol{s}$ is as the latent representation. Autoencoders are trained by introducing a decoder $\mathcal{D}(\boldsymbol{s}) \mapsto \hat{\boldsymbol{I}}$ that maps the latent $\boldsymbol{s}$ back to the input signal $\boldsymbol{I}$. Both encoder $\mathcal{E}$ and decoder $\mathcal{D}$ are trained simultaneously by minimizing the reconstruction loss $\mathcal{L}(\hat{\boldsymbol{I}}, \boldsymbol{I})$. Only the decoder $\mathcal{E}$ is used at test time. This setup forces the low-dimensional latent $\boldsymbol{s}$ to contain sufficient information about the high-dimensional signal $\boldsymbol{I}$ to reconstruct it.

B.2. Contrastive Learning

A contrastive learner learns a state representation by encouraging that some things are similar while others are dissimilar [3, 13, 15]. In particular, contrastive learning uses an objective that encourages the distance between dissimilar pairs to be larger than the distance between similar pairs in the latent space. To learn view-invariance, one would treat images of the same scene state but different camera views to be similar, and images of different scene states but the same camera views to be dissimilar. Similarly to autoencoders, contrastive learners use an encoder $\mathcal{E}(\boldsymbol{I}) \mapsto \boldsymbol{s}$. However, instead of using a decoder, they directly minimize a contrastive loss on the latent embedding $\boldsymbol{s}$.

B.3. Neural Scene Representations

Neural scene representations are coordinate-based neural networks that represent a field [17]. Recently, such neural fields caused a breakthrough in learning implicit 3D representations of scenes, inspired by the seminal work NeRF [11]. By learning a continuous density and radiance function of a 3D scene, NeRF can synthesize novel views of the scene that the model was not trained with. Researchers extended NeRF to make it conditional on a latent code [18], enabling using such a conditional NeRF as the decoder $\mathcal{D}$ in an autoencoding framework. The latent space $\boldsymbol{S} = \{\boldsymbol{s}_1, .., \boldsymbol{s}_N\}$ learned through such a system encodes 3D structure. Li et al. demonstrate using such a 3D neural field encoder for image-based robot control [9].

C. Problem Statement

Given a set of scene states $\mathcal{P} = \{\boldsymbol{p}_1, .., \boldsymbol{p}_M\}$, with multiple observations $\mathcal{I}$ per state, we want to learn a low-dimensional latent space $\boldsymbol{S} = \{\boldsymbol{s}_1, .., \boldsymbol{s}_N\}$ of the given state space. The latent space should encode the scene state corresponding to the image observations $\mathcal{I}$, i.e., distinct scene states should correspond to distinct latent codes. Moreover, the latent space should be invariant to the exact camera poses, i.e., different observations $\mathcal{I}$ of the same scene state $\boldsymbol{p}_i$ should map to the same latent code $\boldsymbol{s}_j$.

D. Methodology

We provide descriptions of the 3D neural field encoder in subsection D.1, the classical autencoder baseline in subsection D.2, and the contrastive learner baseline in subsection D.3.

D.1. 3D Neural Field Encoder

We base our 3D Neural Field Encoder on the one proposed by Li et al. [9]. Specifically, we implement it as described in their paper while filling in implementation details as Li et al. did not provide an implementation.

D.1.1 Neural Radiance Fields

The 3D Neural Field Encoder by Li et al. is based on neural radiance fields (NeRF) [11]. NeRF is a continuous function $F_{\boldsymbol{\theta}} : (\boldsymbol{x}, \boldsymbol{d}) \mapsto (\boldsymbol{c}, \sigma)$ that maps a 3D position $\boldsymbol{x} \in \mathbb{R}^3$ and a 3D viewing direction $\boldsymbol{d} \in \mathbb{R}^3$, $||\boldsymbol{d}|| = 1$ to the corresponding view-dependent color $\boldsymbol{c} \in [0, 1]^3$ and volume density $\sigma \in [0, \inf)$. $\boldsymbol{\theta}$ are learnable parameters. σ represents the probability of hitting a particle while traveling an infinitesimal distance. We can use differential volume rendering to render out an image $\boldsymbol{I}$ at a desired camera pose. This is achieved by marching a camera ray $\boldsymbol{r}(t) = \boldsymbol{x_o} + t\boldsymbol{d}$ through each pixel, where $\boldsymbol{x_o}$ is the camera origin. The RGB color of the image is then calculated as

$$\boldsymbol{I_\theta}(\boldsymbol{r}) = \int_{t_n}^{t_f} w(t)\boldsymbol{c_\theta}(\boldsymbol{r}(t), \boldsymbol{d})\, dt, \tag{1}$$

where t_n, t_f denote the near and far bounds of the ray and

$$w(t) = \exp(-\int_{t_n}^{t_f} \sigma_{\boldsymbol{\theta}}(\boldsymbol{r}(s))\, ds)\sigma_{\boldsymbol{\theta}}(\boldsymbol{r}(t)). \tag{2}$$

$w(t)$ describes the likelihood of a ray terminating at t. The parameters $\boldsymbol{\theta}$ can then be optimized using gradient descent and a mean squared error loss:

$$\mathcal{L}_{rec} = \mathbb{E}_{\boldsymbol{I}\in\mathcal{D}, \boldsymbol{r}\in\mathcal{R}(\boldsymbol{I})} ||\boldsymbol{I}(\boldsymbol{r}) - \boldsymbol{I_\theta}(\boldsymbol{r})||_2^2 \tag{3}$$

where $\mathcal{D}$ is the set of training images and $\mathcal{R}(\boldsymbol{I})$ is the set of rays through each pixel of $\boldsymbol{I}$. Using the ray termination likelihood $w(t)$, it is additionally possible to render out the expected depth image:

$$\boldsymbol{D_\theta}(\boldsymbol{r}) = \int_{t_n}^{t_f} w(t)||\boldsymbol{x_o} - t\boldsymbol{d}||_2\, dt. \tag{4}$$

D.1.2 Conditional Neural Radiance Fields for State Representation Learning

The original NeRF by Mildenhall et al. [11] assumes the scene is static. Consequently, a separate radiance field must be learned for each scene state. Li et al. propose to mitigate this limitation by learning an encoder $\mathcal{E}(\boldsymbol{I}_1, ..., \boldsymbol{I}_K) \mapsto \boldsymbol{s}$ that maps multi-view image observations $\{\boldsymbol{I}_1, ..., \boldsymbol{I}_K\}$ to the latent state $\boldsymbol{s}$. They then learn a function $F_{\boldsymbol{\theta}} : (\boldsymbol{s}, \boldsymbol{x}, \boldsymbol{d}) \mapsto (\boldsymbol{c}, \sigma)$ that differs from the original NeRF function by taking an additional latent code $\boldsymbol{s}$ as an argument. They thus globally condition the neural field on $\boldsymbol{s}$.

In particular, we use a ResNet-18 [4] to extract a feature vector $\boldsymbol{v}$ for each image by taking the output before the pooling layer and passing it through a fully-connected layer. Doing so ensures that spatial information is maintained, which is essential for inferring the scene state. We then concatenate this feature vector $\boldsymbol{v}$ with the flattened cam2world matrix, which represents the camera pose, and process this concatenated vector with a small MLP. We chose two linear layers with ReLU activations for this MLP. To create an image feature for a scene state $\boldsymbol{p}_i$, we average the feature vectors of the individual images $\{\boldsymbol{I}_1, ..., \boldsymbol{I}_K\}$ of the scene state across multiple camera views, pass this average vector through another small MLP, and normalize it to have unit L2 norm. We again chose two linear layers with ReLU activations, with an additional linear layer at the end. We use the resulting vector $\boldsymbol{s}$ as the latent code, representing the scene state $\boldsymbol{p}_i$, and use this latent to condition our neural field. Differently from Li et al., we chose to exclude the view-direction $\boldsymbol{d}$ from $F_{\boldsymbol{\theta}}$, resulting in $F_{\boldsymbol{\theta}} : (\boldsymbol{s}, \boldsymbol{x}) \mapsto (\boldsymbol{c}, \sigma)$. Excluding the view direction results in faster training, which proved essential due to the project time limits. Moreover, our cube, sphere, and background colors are distinct, making them differentiable even under slightly different shadows caused by changes in view direction. We parameterize $F_{\boldsymbol{\theta}}$ using a fully connected neural network with a dense connection as proposed by Li et al. The network architecture is shown in Fig. 2. We use a single network rather than a coarse and a fine network, as proposed in NeRF by Mildenhall et al. We chose this for simplicity and GPU memory concerns, hoping that sampling coarsely is sufficient for our scenes. We use a photometric loss and differential volume rendering to learn the parameters $\boldsymbol{\theta}$ as described in section D.1.1. Similarly to NeRF, we encode the input coordinate $\boldsymbol{x}$ using a frequency $\gamma(\boldsymbol{x})$ encoding as this helps the network represent high-frequency variations in color and geometry. Formally, we use the same frequency encoding as Mildenhall et al.,

$$\begin{aligned}\gamma(\boldsymbol{x}) =&(sin(2^0\pi\boldsymbol{x}), cos(2^0\pi\boldsymbol{x}), ...,\\ &sin(2^{L-1}\pi\boldsymbol{x}), cos(2^{L-1}\pi\boldsymbol{x}))\end{aligned} \tag{5}$$

where $L = 10$. The trigonometric functions are applied element-wise.

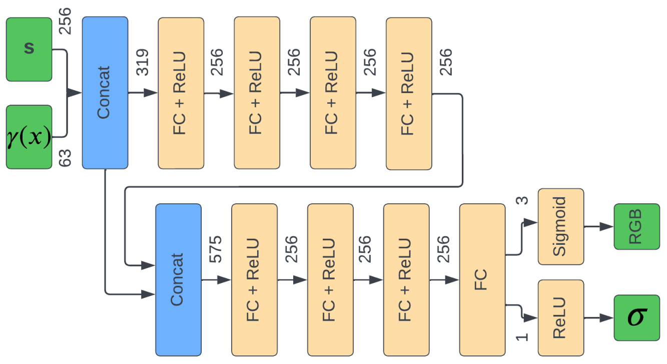

Figure 2. Our neural field decoder architecture. FC denotes fully connected linear layer and Concat denotes concatentation. Numbers denote the size of the vector after each layer. The input is the latent code $\boldsymbol{s}$ and a frequency encoding of the 3D coordinate $\boldsymbol{x}$.

D.1.3 State-Contrastive Loss

Similarly to the time-contrastive loss from Li et al., we use a state-contrastive loss to encourage the encoder to be camera view invariant. Specifically, we encourage features $\boldsymbol{v}$ from the same environment state but different views to be similar to each other, while encouraging features of images from different states to be dissimilar. We use the triplet loss [13],

$$\mathcal{L}_{st} = \max(||\boldsymbol{v}_i^k - \boldsymbol{v}_i^{k'}||_2^2 - ||\boldsymbol{v}_i^k - \boldsymbol{v}_{i'}^k||_2^2 + \alpha, 0), \tag{6}$$

where $\boldsymbol{v}_i^k$ is the feature vector corresponding to a randomly selected anchor observation. $\boldsymbol{v}_i^{k'}$ is a randomly selected feature vector from an image belonging to the same state but a different viewpoint and $\boldsymbol{v}_{i'}^k$ is a feature vector from an image belonging to a different state but same viewpoint as $\boldsymbol{v}_i^k$. α denotes the margin between the similar and dissimilar pairs. We use $\alpha = 2$.

D.1.4 Depth Loss

Depth measurements contain additional information about the scene state. Consequently, we wanted to investigate whether an additional depth loss helps with state representation learning. We render expected depth images $\boldsymbol{D_\theta}$, as described in section D.1.1 and compute a mean squared error loss between the expected depth images $\boldsymbol{D_\theta}$ and the actual depth images $\boldsymbol{D}$:

$$\mathcal{L}_d = \mathbb{E}_{\boldsymbol{D}\in\mathcal{B},\boldsymbol{r}\in\mathcal{R}(\boldsymbol{D})}||\boldsymbol{D}(\boldsymbol{r}) - \boldsymbol{D_\theta}(\boldsymbol{r})||_2^2 \tag{7}$$

where $\mathcal{B}$ is the set of depth images and $\mathcal{R}(\boldsymbol{D})$ is the set of rays through each pixel of $\boldsymbol{D}$. The final loss is a weighted combination of the individual losses: $\mathcal{L} = \mathcal{L}_{rec} + w_{st}\mathcal{L}_{st} + w_d\mathcal{L}_d$.

D.1.5 Training Details

We train our state encoder and neural decoder field end-to-end using an autoencoding architecture. In particular, for each scene state, we randomly sample eight encoding views and two decoding views. We pass the encoding views through the state encoder to obtain the latent $\boldsymbol{s}$. We then use $F_{\boldsymbol{\theta}}$ to render out views at the camera poses corresponding to the decoding views, which we use to compute the reconstruction and depth loss. We chose Adam [6] as our optimizer and initialize all network weights using Kaiming initialization [5]. We train on a V100 GPU for approximately four days.

D.2. Baseline 1: Classical Autoencoder

We chose to use a classical autoencoder as our first baseline (see section B.1 for autoencoding background). In particular, we use an adopted resnet image encoder, as described in section D.1.2 as the encoder and a deconvolutional network as the decoder. Note that we encode and decode a single view instead of multiple views as we do for the neural field decoder. For the deconvolutional network, we use one 2D convolution, followed by six transposed 2D convolutions, followed by another 2D convolution. Each layer, apart from the last layer, is activated using a Leaky ReLU with a negative slope of 0.2. We use a kernel size of one for the first layer and three for the remaining layers. The encoder is trained using an image reconstruction loss:

$$\mathcal{L} = ||\mathcal{D}(\boldsymbol{v}) - \boldsymbol{I}||_2^2 \tag{8}$$

We provide two versions of this baseline; one with and one without an additional contrastive triplet loss, as described in section D.1.3.

D.3. Baseline 2: Contrastive Learner

We chose to use a contrastive learner without autoencoding as our second baseline (see section B.2 for contrastive learning background). Specifically, we encode images into a feature vector $\boldsymbol{v}$ using the adopted resnet encoder as described in section D.1.2. We then directly compute a contrastive loss on the feature vectors $\boldsymbol{v}$ as described in section D.1.3. We provide two versions of this baseline; one with the triplet lss from section D.1.3 and one with an Info-NCE loss [15]. The Info-NCE loss is defined as

$$\mathcal{L} = -\log \frac{\exp\left(\boldsymbol{v}_i^{k\intercal}\boldsymbol{v}_i^{k'}/\tau\right)}{\exp\left(\boldsymbol{v}_i^{k\intercal}\boldsymbol{v}_i^{k'}/\tau\right) + \sum_j \exp\left(\boldsymbol{v}_i^{k\intercal}\boldsymbol{v}_j^{k}/\tau\right)}, \tag{9}$$

where τ is a scalar temperature that controls the loss' smoothness. As for the triplet loss, the loss aims to make observations from the same state i but different viewpoints k' closer in the state space, while making observations from different states j but same viewpoint k farther in the state space. Note that the contrastive baselines are only required to encode that different scene states should map to different latent codes. The latents do not need to contain sufficient information to reconstruct an image of the scene.

E. Experimental Results

E.1. Data Generation

We generate synthetic training data using Drake [14]. In particular, we setup the planar cube environment shown in Figure 1 and sample states uniformly across the entire state space, where neighboring states are one box/sphere width apart. This results in 6480 states. For each state, we record 24 training views and 11 evaluation views as shown in Figure 3. We additionally record the camera extrinsics and intrinsics for each view.

E.2. Evaluation Metrics

Our goal is to learn a low-dimensional latent space $\boldsymbol{S} = \{\boldsymbol{s}_1, .., \boldsymbol{s}_N\}$ that should encode the scene state corresponding to the image observation $\mathcal{I}$ and be viewpoint invariant. We compare the learned latent spaces using t-SNE [16], which is a dimensionality

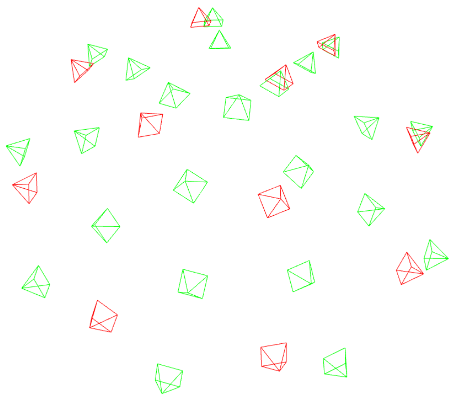

Figure 3. Camera frustums, showing the cameras' field of view. The camera views used for training are shown in green and the ones used for evaluation are shown in red. Notice that there are big differences between the training and evaluation views.

reduction algorithm. Specifically, t-SNE models high-dimensional latents $\boldsymbol{s}$ by two dimensional vectors $\boldsymbol{t}$, such that similar latents are modeled by nearby vectors and dissimilar latents are modeled by distant vectors with higher probability. We consider the learned state space to be of high quality if different viewpoints of the same state are closer to each other in the t-SNE space than views of different states. We evaluate this closeness both qualitatively by plotting a random sample of states with multiple viewpoints and quantitatively by comparing the scene states of nearest neighbors in the t-SNE space. If a vector in the t-SNE space has neighbors that belong to the same scene state, then the learned latent space contains the desired closeness information. We perform the nearest neighbor search on the t-SNE space rather than the original latent space due to computational constraints.

E.3. Findings

Fig 6 shows the results of our nearest neighbor metric for our 3D neural decoders and the baselines on the evaluation camera viewpoints. It can be observed that all models learn something about view-invariance as the probability of randomly getting a closest neighbor correct is 0.015%. The 3D decoders ("nerf_ae") outperform the classical auto-encoders ("vanilla_ae"). Moreover, the 3D decoder with the contrastive loss ("nerf_ae_ct") outperforms the contrastive learner with the same contrastive loss ("ct_triplet"), while the 3D decoder without the contrastive loss ("nerf_ae") achieves similar performance to the contrastive learner ("ct_triplet"). This demonstrates the potential for improving view-invariant state representation learning with 3D neural scene representations. The contrastive learner with the Info-NCE loss ("ct_info_nce") still outperforms the 3D decoders. However, it would be possible that replacing the triplet loss in "nerf_ae_ct" with an Info-NCE loss would outperform this baseline as this is what we observed for "nerf_ae_ct" and "ct_triplet". The results additionally show that adding depth supervision to 3D neural decoders decreases performance. This is the case for both "nerf_ae_depht" and "nerf_ae_ct_depht" as compared to their versions without the depth loss "nerf_ae" and "nerf_ae_ct". In general, both contrastive learning and 3D neural scene representations seem to be a good idea for learning viewpoint-invariant state representations. Combining these two paradigms can outperform models that only use one of them.

Our qualitative results confirm the nearest neigbor results.

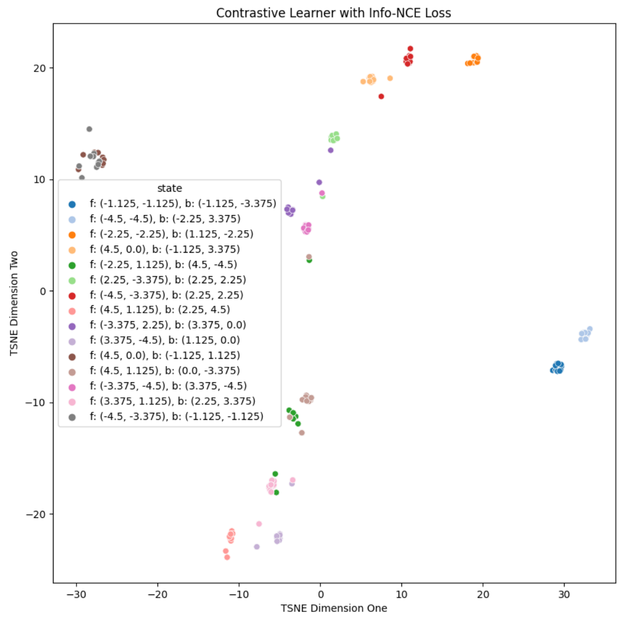

Figure 4. t-SNE visualization of the contrastive learner with the Info-NCE loss. 15 random states and their camera viewpoints in the evaluation dataset are shown. Latents belonging to the same state but different viewpoints are shown with the same color. Notice that different viewpoints of the same state are clustered together.

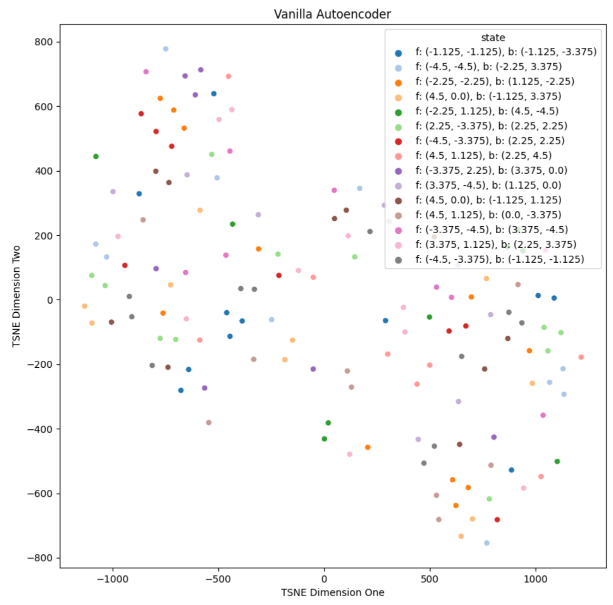

Figure 5. t-SNE visualization of the classical autoencoder. 15 random states and their camera viewpoints in the evaluation dataset are shown. Latents belonging to the same state but different viewpoints are shown with the same color. Notice that different viewpoints of the same state are not clustered as nicely as for the contrastive learner with the Info-NCE loss (Fig. 4).

Fig. 4 shows the t-SNE plot of 15 randomly samples states for the Info-NCE contrastive learner and Fig. 5 shows the one for the same states for the classical autoencoder. It is clearly visible that the contrastive learner learns a state space in which latents belonging to the same state but different viewpoints are closer to each other than latents from different states. The state space from the classical autoencoder is significantly more mixed up.

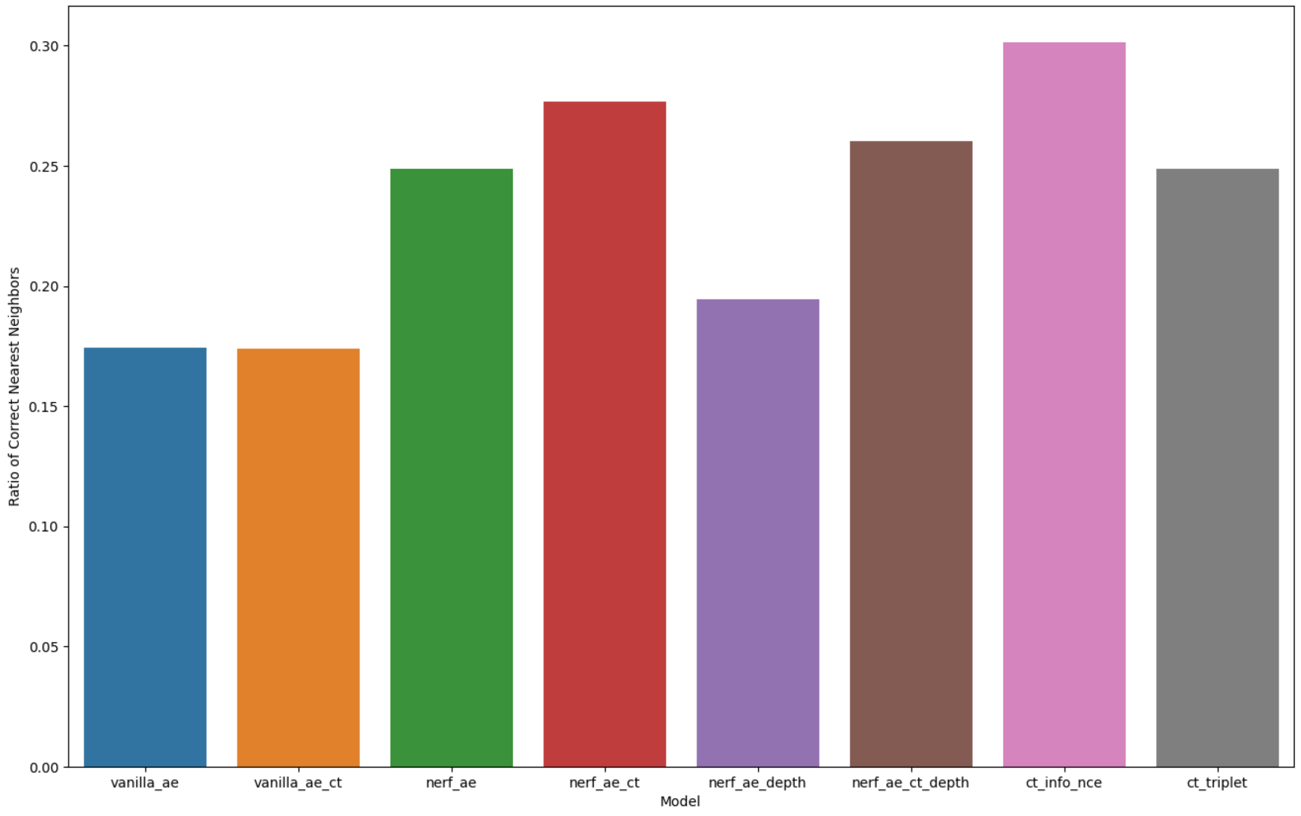

Figure 6. The ratio of correctly identified nearest neighbors in the t-SNE space for the evaluation camera viewpoints. The state space contains 6480 states with 11 views each, resulting in 71280 latents. For each latent, we query the 11 nearest neighbors and report the ratio of of the neighbors that belong to the same state as the query latent.

F. Future Work

We showed that a 3D neural field decoder with a contrastive loss can outperform a contrastive learner with the same contrastive loss. However, we did not produce results for a 3D decoder with an Info-NCE loss. A fruitful direction of future research would be to investigate whether this behavior holds in the general case by testing whether a 3D decoder with an Info-NCE loss can outperform a contrastive learner with that loss.

We explored state representation learning for a planar pushing task, where the state of the scene is a 2D box and a 2D finger position. Consequently, it is a 2D rather than a 3D task, such as object stacking. In the future, it would be interesting to explore how 3D neural field decoders perform on 3D tasks. A particularly interesting task would be one with a 3D nature and occlusions that prevent a single image from containing sufficient information to identify the entire scene state.

G. Conclusion

In this project, we explored whether 3D neural scene representations are the right choice for view-invariant state representation learning. We found that there is potential in improving view-invariant state representation learning with such 3D representations by comparing them with multiple baselines. Noticeably, our results show that combining 3D representations with contrastive learning is particularly effective. Exciting future work would be exploring whether combining 3D representations with contrastive learning outperforms the individual paradigms in the general case.

References

[1] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives, 2014. 1

[2] Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion, 2023. 1

[3] Adam Foster, Rattana Pukdee, and Tom Rainforth. Improving transformation invariance in contrastive representation learning, 2021. 1

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. 1, 2

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, 2015. 3

[6] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. 3

[7] Timoth'e Lesort, Natalia Diaz-Rodriguez, Jean-Francois Goudou, and David Filliat. State representation learning for control: An overview. *Neural Networks*, 108:379–392, dec 2018. 1

[8] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies, 2016. 1

[9] Yunzhu Li, Shuang Li, Vincent Sitzmann, Pulkit Agrawal, and Antonio Torralba. 3d neural scene representations for visuomotor control. *arXiv preprint arXiv:2107.04004*, 2021. 1, 2

[10] Yang Liu, Zhaoyang Lu, Jing Li, and Tao Yang. Hierarchically learned view-invariant representations for cross-view action recognition. *IEEE Transactions on Circuits and Systems for Video Technology*, 29(8):2416–2430, aug 2019. 1

[11] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis, 2020. 2

[12] Faegheh Sardari, Björn Ommer, and Majid Mirmehdi. Unsupervised view-invariant human posture representation, 2021. 1

[13] Florian Schroff, Dmitry Kalenichenko, and James Philbin. FaceNet: A unified embedding for face recognition and clustering. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jun 2015. 1, 3

[14] Russ Tedrake and the Drake Development Team. Drake: Model-based design and verification for robotics, 2019. 3

[15] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding, 2019. 1, 3

[16] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008. 3

[17] Yiheng Xie, Towaki Takikawa, Shunsuke Saito, Or Litany, Shiqin Yan, Numair Khan, Federico Tombari, James Tompkin, Vincent Sitzmann, and Srinath Sridhar. Neural fields in visual computing and beyond, 2022. 2

[18] Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. pixelNeRF: Neural radiance fields from one or few images. In *CVPR*, 2021. 2