Sequential Contrastive Learning to Master Effective Representations For Reinforcement Learning and Control

Ali Younes[a], Aleksandr I. Panov[b,c]

[a]Bauman Moscow State Technical University, Moscow, Russia
[b]Moscow Institute of Physics and Technology, Moscow, Russia
[c]Federal Research Center "Computer Science and Control" of the Russian Academy of Sciences,Moscow, Russia

Abstract
State estimation is an essential part of any control system. A perception system estimates a representation of the states using sensors data. Recently, an increasing interest in exploiting machine learning techniques to learn state representation. Representation learning allows estimating states in real-world scenarios. We are interested in learning representation from RGB images extracted from videos. And use the state representation to compute cost/reward suitable for control and reinforcement learning. We propose a method in which the user has to provide just a couple of videos demonstrating the task. Our approach uses a sequential contrastive loss to learn a latent space mapping, and descriptors of the task-related objects. Our framework serves robotics control scenarios, especially model-based reinforcement learning algorithms. The resulted representation eliminates the need for engineered reward functions or any explicit access to positioning systems, aiming to improve the applicability of learning to control physical systems. Our framework allows for reducing the learning time and working with low-resource scenarios.

Keywords
Model-based Reinforcement Learning, Learning Representation, Sequential Contrastive Learning

1. Introduction

Optimal control[Zho96, Cam13] deals with finding the best possible control action by optimizing a criterion. The criterion is a cost function that is a function to the state and the control. Optimal control is beneficial in the presence of constraints on the control variables or the states. In optimal control, we use the system equation to predict the next state according to the current state and the executed control action, which means a need for a state estimation system, to find the current state.

Reinforcement learning [Sut98] describes the system as a Markov decision process. An agent interacts with an environment by choosing actions that maximize a reward. Actions are produced by a policy, which is a function of the environment state. In the Markov decision process

Russian Advances in Artificial Intelligence: selected contributions to the Russian Conference on Artificial intelligence (RCAI 2020), October 10-16, 2020, Moscow, Russia
ay20-5-1994@hotmail.com (A. Younes); pan@isa.ru (A.I. Panov)
0000-0002-9747-3837 (A.I. Panov)

notion, the sensor data defines the observations, which used to estimate the states. Based on the states, we compute the reward function.

In simple cases, the state is proportional to the sensor data. For more complex cases, states computed using the dynamical model (forward kinematic model for the position of the end-effector [Jaz10], odometry for a mobile robot). In compound tasks, external sensors are used (positioning systems[Mer17], RGBD cameras[Sch15]) to get a relative position of an object in the surroundings. Forming representative states in such cases needs tremendous effort and suffers from inaccuracy. This fact leads to exploiting machine learning to generate relevant state representations, to be used in control and reinforcement learning. We will concentrate on the learning representations from visual input to control physical systems.

Feature extraction from images [Nix19] is a well-studied topic in computer vision literature, which aims to extract features that characterize an image. Deep learning allows automatically learning the extraction of low-dimensional features from high-dimensional images. State representation learning [Les18] is a case of feature learning that has additional requirements to encode information about the time and interactions with the environment.

End-to-end learning to control [Lev16] maps observations to actions, by feeding observation to a deep neural network. A convolutional part is essential when the observations are images (learning from pixels), the output of this convolutional part is a low-dimensional vector. This vector representation is not interpretable and can't be used to compute the cost/reward. End-to-end learning needs an external reward signal. This type of learning suffers from a long training time as the representation is learned from scratch.

Autoencoders [Bal12] and variational autoencoders [Pu16] can be applied to learn low-dimensional representations without supervision. Autoencoders used to aid visuomotor policy learning. Special types of encoders allow controlling the distribution of features in the latent space [Mak15], while others provide information about the spatial information of the states [Fin16]. Autoencoders reduce the training time and give task-agnostic representation. Their problem lies in the difficulties of using the learned features to compute the cost/reward. The usual solution is comparing states with given goal images to compute a sparse reward.

Time-Contrastive Network [Sem18, Sem17, Dwi18] uses a self-supervised learning approach to learn representations entirely from unlabeled videos. TCN makes use of a triplet loss [Soh16] combined with a multi-view metric to ensure that the features disentangled in the latent space following the task progress. The reward /cost after learning is the Euclidean distance between the current state and the target state in the latent space. Building on the TCN, we are proposing a sequential training procedure.

Dense object nets [Flo18] use a pixel contrastive loss [Sch16] to learn descriptors of an image, these descriptors provide information about the objects in the scene. The position of the descriptors can serve as features in visuomotor policy learning [Flo19]. Unlike DON, we won't use RGBD cameras, we will learn from RGB images. We are proposing a triplet pixel loss to learn descriptors of the task-related object in a self-supervised way.

Contributions. Our primary contribution is (1) a novel formulation for self-supervised representation learning for low-resource scenarios (RGB cameras, single GPU). (2) our model will output an embedding of the states in the latent space, alongside dense descriptor image of the task-related objects. (3) we are proposing a new sequential contrastive loss and a contrastive pixel loss. The framework and the experiments is beneficial for robotics manipulation tasks

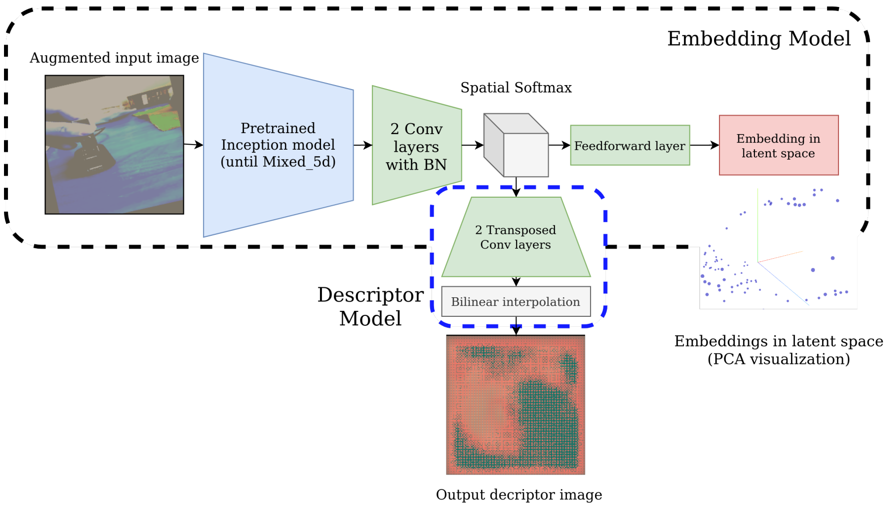


Figure 1: The architecture of the embedding and the descriptor models. The blue block is a pre-trained part fro the inception model (frozen), Green blocks are layers with trainable variables, white blocks are operators. The embedding model takes an augmented image as an input, outputs an embedding in the latent space. The descriptor uses the spatial features of the image from, to output a descriptor image in which task-related objects has distinguished descriptors.

(similar to pig-in-hole insertion and pick-and-place).

2. Methodology

2.1. The model

Consider we have an input image $x \in \mathbb{R}^{W\times H\times 3}$. The model will have two outputs; a descriptor image $y_d \in \mathbb{R}^{W\times H\times D}$ and an embedding of the image in a latent space $y_e \in \mathbb{R}^{E}$, W, H are the width and the height of the image, D is the depth of the descriptor, E the size of the embedding vector . The base CNN network is taken from the inception model [Sze17] pre-trained on ImageNet [Den09], the parameters of this part is frozen and won't be re-trained. We add two additional convolutional layers, followed by a spatial softmax layer. Spatial features ($S_{sm}(x)$) are the output of this part. A fully connected layer is added to get the embedding of the input image $y_e = f_\theta(x)$. The parameter to train embedding model θ are the parameters of the convolutional layers, and the parameters of the fully connected layer.

Additional descriptor model takes the spatial feature S_{sm} from the embedding model as input. The descriptor model consists of two transposed convolutional layers, followed by a bilinear interpolation operation to upscale the output to the same of the input image. The output of the descriptor model is a descriptor image $y_d = f_\phi(S_{sm})$, where $S_s m(x)$ is the output of the spatial softmax operation from a pretrained embedding model.

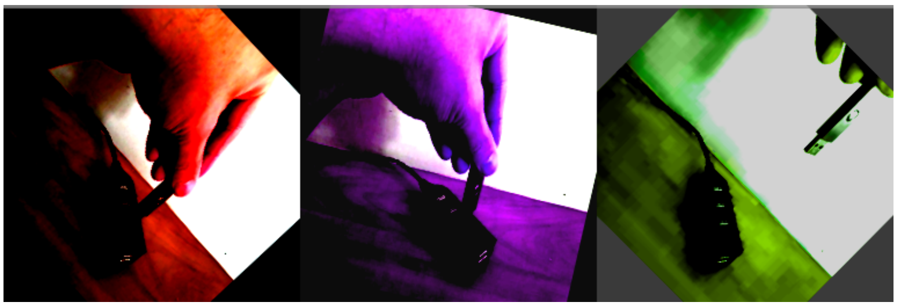


Figure 2: Example of a triplet, the anchor, positive and negative images. All images are augmented by a random coloring and random rotation.

2.2. Dataset formulation

The dataset is formed by extracting images from video files and form triplets for learning. In contrastive learning, the process starts by sampling an image from a video, this image is called anchor, a positive range is defined as all the images that far from the anchor by less than a positive margin, the negative margin is the rest of the images. one positive and one negative images are sampled from the positive and negative ranges respectively. The triplet consists of an anchor x_i^a, a positive image x_i^p and a negative image x_i^n.
All images normalized and augmented before feeding them to the network. The normalization is need because the pretrained part trained on normalized images. The augmentation was done by using a color jitter (random coloring) and random rotation. The augmentation is useful for transfer learning, and to avoid over-fitting to the training dataset.

In our method - Sequential contrastive learning, we introduced a scheduled update for the positive margin, and instead of having just one negative margin, we used near and far negative margins. All margins will be updated during the training, alternating between tight positive range, and tight, not-that-far negative range to wider positive range, and wide, for negative range. Our goal is to encourage disentangling near embeddings before separating them into clusters, which gives better distribution in the latent space.

2.3. Sequential Time Contrastive Loss

Given a triplet of images; an anchor x_i^a, positive image x_i^p, and negative image x_i^n, the time-contrastive loss is the loss tries to ensure that the anchor and the positive images are closer to each other in the latent space than the negative image. i.e. the aim is to learn an embedding such that:

$$||y_e(x_i^a) - y_e(x_i^p)||_2^2 + \alpha < ||y_e(x_i^a) - y_e(x_i^n)||_2^2$$

where α is a margin that is enforced between x_i^p and x_i^n. I.e. trying to disentangle the representations in accordance with the task progress (time).

In our method - Sequential contrastive learning, we are proposing using a scheduled update of the margin between x_i^p and x_i^n. The update of the loss margin will be synchronized with the update of the positive and negative margins of the sampling from the dataset. The goal of this update is to encourage making the distance in the latent space between the clusters of similar images according to the task progress. At the same time, it will ensure disentangling embeddings inside the clusters.

The sequential time contastive loss is defined as:

$$L_{stcl} = min(0, \alpha_i + distance(anchor, positive) - distance(anchor, negative))$$

$$L_{stcl} = min(0, \alpha_i + ||y_e(x_i^a) - y_e(x_i^p)||_2^2 - ||y_e(x_i^a) - y_e(x_i^n)||_2^2)$$

2.4. Sequential Pixel-wise Contrastive Loss

Pixel-wise contrastive loss is defined depending on the correspondences between the pixels of a pair of RGB images $I_a, I_b \in \mathbb{R}^{W \times H \times 3}$. In Dense Object Networks [Flo18] they used depth information, a pixel $u_a \in I_a$ matches a pixel $u_b \in I_b$ if they correspond to the same vertex of the dense 3D reconstruction. While this assumption is accurate and gives satisfying results, we claim that the 3D reconstruction works but it is laborious.

In our method, we won't use depth information, only RGB cameras will be used. Our workaround is to making use of the spatial features learned in the embedding model training process, and define a sequential pixel-wise contrastive loss.

Given two RGB images $x_1, x_2 \in \mathbb{R}^{W \times H \times 3}$ (we will sample a triplet from our dataset, and use the anchor and the negative images as x_1, x_2 respectively). We will feed each of the images to a trained embedding model, instead of using the output of the fully connected layer, we use the output of the softmax layer a.k.a. spatial features. The results are two spatial features $S_{sm}(x_1), S_{sm}(x_2)$, by feeding each of them to the descriptor model we will get two descriptor images $y_{d1} = y_d(S_s m(x_1)), y_{d2} = y_d(S_s m(x_2))$

To find the pixel-wise contrastive loss, we extracted the indices of the features from each image $f_1 = edges(x_1), f_2 = edges(x_1)$, the features in our case was the edges detected by the Canny edge detector. Pixels in the output descriptor images correspond to the indices will be used to compute the positive distance $||y_{d1}[f_1] - y_{d2}[f_2]||_2^2$. Negative points f_{neg} will be sampled randomly from the second image, will be used to compute the negative distance $||y_{d1}[f_1] - y_{d2}[f_{neg}]||_2^2$. The pixel-wise contrastive loss will be defined:

$$L_{spcl} = min(0, \alpha_i + distance(feat_1, feat_2) - distance(feat_1, feat_{neg}))$$

$$L_{spcl} = min(0, \alpha_i + ||y_{d1}[f_1] - y_{d2}[f_2]||_2^2 - ||y_{d1}[f_1] - y_{d2}[f_{neg}]||_2^2)$$

Where α_i is a scheduled margin, the sequential update of the margin will encourage the descriptor model to differentiate the close images from far images, which leads to better descriptors for task-related objects.

2.5. Training the whole system

Firstly we have to define the margins (in the following we present the best margins according to our experiments):

L- the number of frames extracted from each video
pos_margins = [L/4, L/5, L/10, L/20, 2]
neg_margin_near = [L/3, L/4, L/5, L/10, L/20]
neg_margin_far = [L, L/2, L/3, L/4, L/5]
Loss margins
STCL_margins = [12.5, 10, 7.5, 3.5, 1]
SPCL_margins = [25, 15, 10, 5, 2]

The function update margins() updates the sampling margins of the dataset (pos margins, neg margins near and neg margins far), and the values of α variables for each of the losses (STCL and SPCL margins).

Training of the embedding model, starts by sampling a triplet of augmented images from the dataset, feeding them to the model, and use the output to compute the sequential time contrastive loss, the loss then is used to update the parameters of the model. The margins are updated every while.

```
1 Sample triplets from the dataset $x_i^a, x_i^p, x_i^n \sim D$
2 Feed images to the model $y_e^a, y_e^p, y_e^n = f_\theta(x_i^a), f_\theta(x_i^p), f_\theta(x_i^n)$
3 Compute the loss $L_{stcl} = min(0, \alpha_i + ||y_e^a - y_e^p||_2^2 - ||y_e^a - y_e^n||_2^2)$
4 Update the parameters $\theta \longleftarrow argmin_\theta(L_{stcl})$
5 Every while : update margins()
6 Go to step 1
```

Algorithm 1: Training the embedding model

The same procedure is used to train the descriptor model, the difference is the use of the trained embedding model to extract the spatial features, and the use of the sequential pixel-wise contrastive loss.

```
1 Sample from the dataset $x_1,, x_2 \sim D$
2 Get the spatial features $S_{sm1}, S_{sm2} = f_\theta(x_1), f_\theta(x_2)$ Feed the spatial features to the model
    $y_{d1}, y_{d2} = f_\phi(S_{sm1}), f_\phi(S_{sm2})$
4 Compute the loss $L_{spcl} = min(0, \alpha_i + distance(feat_1, feat_2) - distance(feat_1, feat_{neg})$
5 Update the parameters $\phi \longleftarrow argmin_\phi(L_{spcl})$
6 Every while : update margins()
7 Go to step 1
```

Algorithm 2: Training the descriptor model

3. Experiment

Self-supervised training makes it easy to test our system. The user has to collect videos (possibly using smartphones) demonstrating the task that we need to learn its representations. We don't have any constraints on the video length or size, in our case, we collected 3 videos of a USB insertion task. The demonstration was done by human hand, we recommend shooting videos from many viewpoints, and if possible add a video of a robotic manipulator moves

Table 1
The margins used in the sequential contrastive learning experiment

	α_1	α_2	α_3	α_4	α_5
pos margins	25	20	10	5	2
near negative margins	50	25	20	10	5
far negative margins	100	50	25	20	10
time contrastive margins	12.5	7.5	5	2.5	1
pixel-wise contrastive margins	50	25	20	10	5

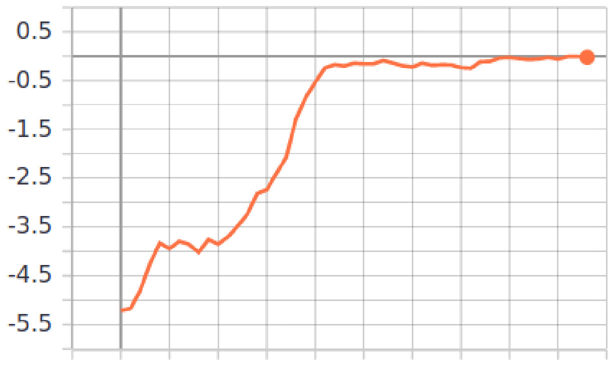


Figure 3: The reward function for the evaluation video - see the project web-page

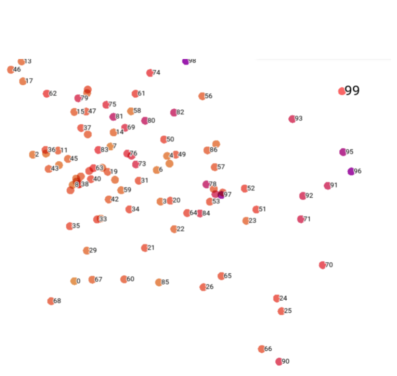
randomly to train the descriptor model.

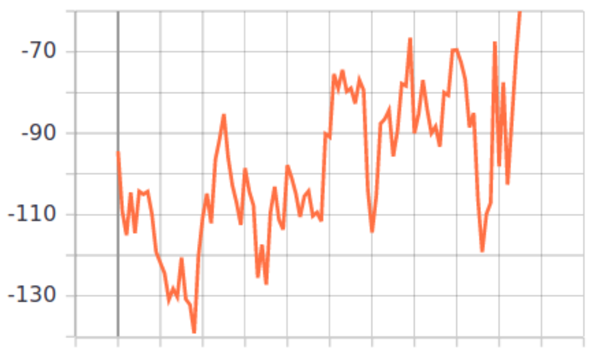
When running the framework, 100 frames will be extracted from each video. Frames will be resized and saved to a list. The dataset class will sample an anchor, positive and negative images (in accordance with the margins), normalize, and augment them when the data loader asks for samples. The hardware used for training is a PC with a single GPU NVIDIA GTX 1050Ti. The usage of memory in our case was less than 1 GB.

The embedding model consists of a pre-trained inception model (until Mixed_5d layer), followed by a convolutional layer with 100 filters, a batch normalization layer, second convolutional layer (kernel size 3, stride 1) with filters with a number equal to the size of the spatial softmax layer, followed by second batch normalization layer. After that, we have a spatial softmax layer, and lastly a fully connected layer with output size equal to the desired embedding size (in our experiment was 32).

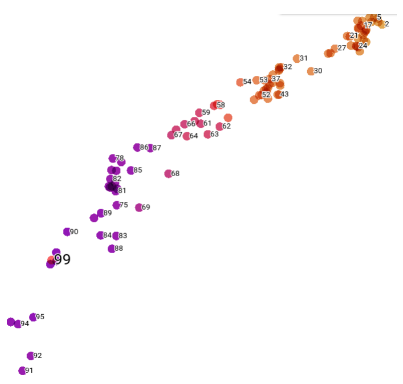
The descriptor model consists of a transposed convolution layer (kernel size 5, stride 2), its input has the same number of the channel of the size of the spatial features, the output has 6 channels, followed by another transposed convolutional layer with 3 output channel (to be visualized as an RGB image - we can use other number to learn densely descriptors). Lastly, we use a bilinear interpolation operation to scale up the result to the same size as the input image.

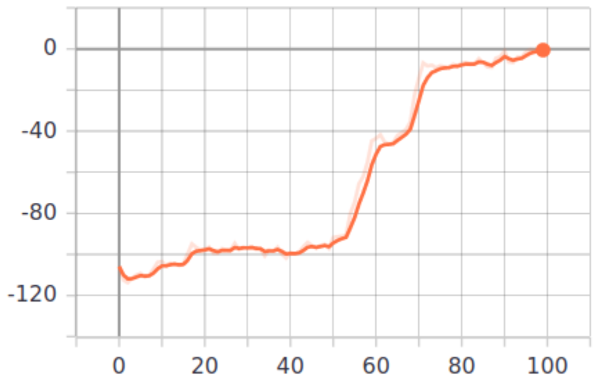
The sequential learning is performed by updating the margins every-while during training. Empirical results led to choosing dataset margins (positive and negative margins) proportional to the number of frames extracted from each video. Margins of the sequential contrastive


(a) Latent space before training

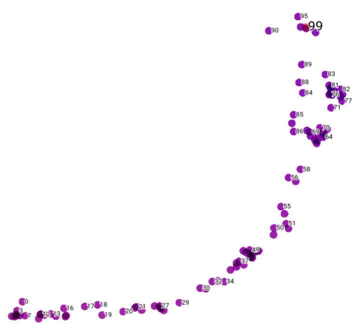


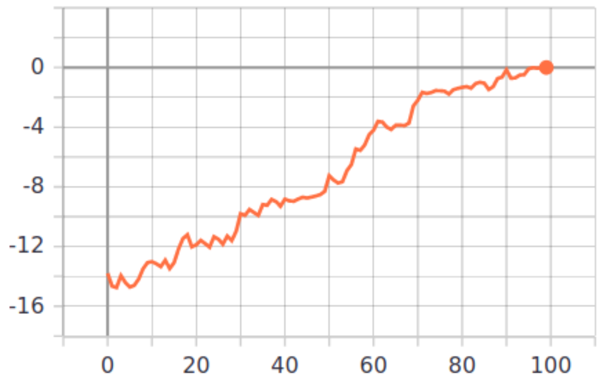
(b) Reward before training


(c) Latent space - no sequential training



(d) Reward - no sequential training


(e) Latent space - sequential learning



(f) Reward - sequential learning

Figure 4: Latent space visualization of a video from the training dataset (100 video frames), (a) before training. (c) after time contrastive training. (e) after using the sequential updates. Better intra-class distribution when using sequential learning. The reward/cost function corresponding to each latent space (b) before training, random distribution led to a bad reward function. (d) after time contrastive training, better distribution, but neighbor images have similar rewards (f) after using the sequential updates, better monotonically increasing function, could be used in RL and control.(The range of the reward function depends on the margins α, and sequential learning we have a compact range which is better)

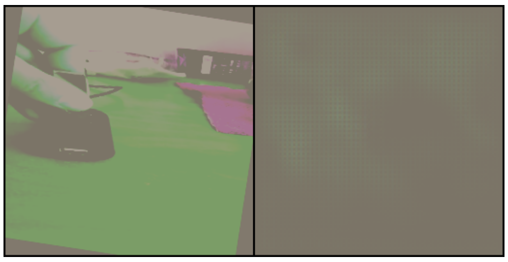
losses chosen proportional to the gap between the positive and the near negative margins of the dataset. Margins of the sequential pixel-wise contrastive loss should be chosen big enough to enforce contrast. Margins associated with the best performance during our experiments listed in Table 1.

The embedding model was trained for 100 epochs (10000 triplets), margins start to be updated after 25 epochs (2500 triplets). and then, all margins of the losses and the database updated every 5 epochs (500 triplets). Alongside the visualization of the latent space (embeddings) in Figure 4, we have plotted a reward function depends on the distance to the target image in latent space:

$$r(x_i) = -\|y_e(x_i) - y(x_{target})\|_2^2$$

For each video file, the reward function should be monotonically increasing to zero, smoother function means better performance. To judge the benefit of using sequential learning, we plotted the latent space and the reward function before using the sequential updates. We can notice, the sequential learning led to better intra-class distribution in the latent space, i.e. better rewards.

To validate the trained model, we have tested it on a video from outside the training dataset, to harden the evaluation, we used a different USB flash for the new video. Plotting the reward function for the evaluation video[1] (Fig. 3) shows a reward function with suitable values.


(a) a

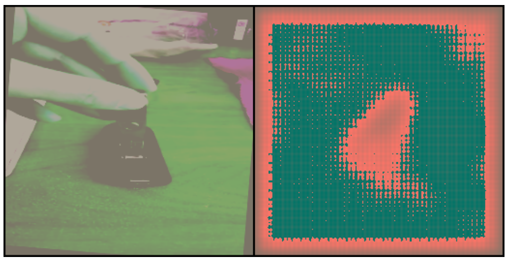

(b) b

Figure 5: Descriptor images (a) before training, No information in the descriptor image (b) after training, task-related objects have unique descriptor other than background objects

Training the descriptor model took 20 minutes (10000 triplets = 30000 images). In fig.5 we have visualized the descriptor image at the beginning and at the end of the training procedure. the model can distinguish the task-related parts and give them distinguishable descriptors even when the image is augmented, which gives a sign of the robustness of our method.

The visualization of descriptor images is a proof-of-concept. We need additional processing to extract the reward from them, which is out of coverage in this paper.

4. Conclusion

We have presented a self-supervised representation learning framework for robotics manipulation tasks. The framework makes use of sequential and contrastive learning to master a

[1]The training and evaluation videos are available on the project page: https://alonso94.github.io/SCL/

better distribution of the images embeddings in the latent space. The distribution corresponds to the task progress. Also, pixel-wise training was used to learn the descriptor representation of the image, the resulted model was able to concentrate on the task-related object even when working with augmented images (color changing and random rotation).

The training time for a task is relatively short (around one hour), and the results could be improved by further learning. We have demonstrated by experiments the ability to learn reasonable results during these time, which gives a good usability feature to the framework.

The framework provides a promising way to run robotics experiments, as the user has to collect some videos (possibly with a smartphone) demonstrating the task. The framework better to be integrated with model-based reinforcement learning, or other optimal control algorithms to provide fully automated experiments.

Open-source code is made available for reproducibility and validation
https://alonso94.github.io/SCL/.

Acknowledgments

The reported study was supported by RFBR, research Project No. 18-29-22027.

References

[Zho96] Zhou, K., Doyle, J. C., & Glover, K. (1996). Robust and optimal control (Vol. 40, p. 146). New Jersey: Prentice hall.

[Cam13] Camacho, E. F., & Alba, C. B. (2013). Model predictive control. Springer Science & Business Media.

[Sut98] Sutton, R. S., & Barto, A. G. (1998). Introduction to reinforcement learning (Vol. 135). Cambridge: MIT press.

[Jaz10] Jazar, R. N. (2010). Theory of applied robotics: kinematics, dynamics, and control. Springer Science & Business Media.

[Mer17] Merriaux, P., Dupuis, Y., Boutteau, R., Vasseur, P., & Savatier, X. (2017). A study of vicon system positioning performance. Sensors, 17(7), 1591.

[Sch15] Schwarz, M., Schulz, H., & Behnke, S. (2015, May). RGB-D object recognition and pose estimation based on pre-trained convolutional neural network features. In 2015 IEEE international conference on robotics and automation (ICRA) (pp. 1329-1335). IEEE.

[Nix19] Nixon, M., & Aguado, A. (2019). Feature extraction and image processing for computer vision. Academic press.

[Les18] Lesort, T., Díaz-Rodríguez, N., Goudou, J. F., & Filliat, D. (2018). State representation learning for control: An overview. Neural Networks, 108, 379-392.

[Lev16] Levine, S. (2016). Deep Learning for Robots: Learning From Large-Scale Interaction. Google Research Blog, Março.

[Bal12] Baldi, P. (2012, June). Autoencoders, unsupervised learning, and deep architectures. In Proceedings of ICML workshop on unsupervised and transfer learning (pp. 37-49).

[Pu16] Pu, Y., Gan, Z., Henao, R., Yuan, X., Li, C., Stevens, A., & Carin, L. (2016). Variational autoencoder for deep learning of images, labels and captions. In Advances in neural information processing systems (pp. 2352-2360).

[Mak15] Makhzani, A., Shlens, J., Jaitly, N., Goodfellow, I., & Frey, B. (2015). Adversarial autoencoders. arXiv preprint arXiv:1511.05644.

[Fin16] Finn, C., Tan, X. Y., Duan, Y., Darrell, T., Levine, S., & Abbeel, P. (2016, May). Deep spatial autoencoders for visuomotor learning. In 2016 IEEE International Conference on Robotics and Automation (ICRA) (pp. 512-519). IEEE.

[Sem18] Sermanet, P., Lynch, C., Chebotar, Y., Hsu, J., Jang, E., Schaal, S., ... & Brain, G. (2018, May). Time-contrastive networks: Self-supervised learning from video. In 2018 IEEE International Conference on Robotics and Automation (ICRA) (pp. 1134-1141). IEEE.

[Sem17] Sermanet, P., Lynch, C., Hsu, J., & Levine, S. (2017, July). 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.

[Dwi18] Dwibedi, D., Tompson, J., Lynch, C., & Sermanet, P. (2018, October). Learning actionable representations from visual observations. In 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (pp. 1577-1584). IEEE.

[Soh16] Sohn, K. (2016). Improved deep metric learning with multi-class n-pair loss objective. In Advances in neural information processing systems (pp. 1857-1865).

[Flo18] Florence, P. R., Manuelli, L., & Tedrake, R. (2018). Dense object nets: Learning dense visual object descriptors by and for robotic manipulation. arXiv preprint arXiv:1806.08756.

[Sch16] Schmidt, T., Newcombe, R., & Fox, D. (2016). Self-supervised visual descriptor learning for dense correspondence. IEEE Robotics and Automation Letters, 2(2), 420-427.

[Flo19] Florence, P., Manuelli, L., & Tedrake, R. (2019). Self-Supervised Correspondence in Visuomotor Policy Learning. IEEE Robotics and Automation Letters.

[Sze17] Szegedy, C., Ioffe, S., Vanhoucke, V., & Alemi, A. A. (2017, February). Inception-v4, inception-resnet and the impact of residual connections on learning. In Thirty-first AAAI conference on artificial intelligence.

[Den09] Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., & Fei-Fei, L. (2009, June). Imagenet: A large-scale hierarchical image database. In 2009 IEEE conference on computer vision and pattern recognition (pp. 248-255). Ieee.