# Supplementary Material for Unsupervised Layered Image Decomposition into Object Prototypes

Tom Monnier[1] Elliot Vincent[1,2] Jean Ponce[2,3] Mathieu Aubry[1]
[1]LIGM, École des Ponts, Univ Gustave Eiffel, CNRS
[2]Inria and DIENS (ENS-PSL, CNRS, Inria) [3]Center for Data Science, New York University

In this supplementary document, we provide quantitative semantic segmentation results (Section A), analyses of the model (Section B), training details (Section C) and additional qualitative results (Section D).

## A. Quantitative semantic segmentation results

We provide a quantitative evaluation of our approach in an unsupervised semantic segmentation setting. We do not compare to state-of-the-art approaches as none of them explicitly model nor output categories for objects. We argue that modeling categories for discovered objects is crucial to analyse and understand scenes, and thus advocate such quantitative semantic evaluation to assess the quality of any object-based image decomposition algorithm.

**Evaluation.** Motivated by standard practices from supervised semantic segmentation and clustering benchmarks, we evaluate our unsupervised object semantic segmentation results by computing the mean accuracy (mACC) and the mean intersection-over-union (mIoU) across all classes (including background). We first compute the global confusion matrix on the same 320 images used for object instance segmentation evaluation. Then, we reorder the matrix with a cluster-to-class mapping computed using the Hungarian algorithm [10]. Finally, we average accuracy and IoU over all classes, including background, yielding mACC and mIoU.

**Results.** Our performances averaged over 5 runs are reported in Table 1. Similar to our results for object instance segmentation, we filter an outlier run out of 5 for Multi-dSprites based on its high reconstruction loss compared to other runs ($1.93 \times 10^{-3}$ against $\{1.51, 1.49, 1.52, 1.57\} \times 10^{-3}$). For Tetrominoes and Multi-dSprites, our method obtains strong results thus emphasizing that our 2D prototype-based modeling is well suited for such 2D scene benchmarks. On the contrary for CLEVR6, there is still room for improvements. Although we can distinguish the 6 different categories from discovered sprites, such performances suggest that our model struggles to accurately transform the sprites to match the target object instances. This is expected since we do not explicitly account for neither 3D, lighting nor material effects in our modeling.

Table 1: **Multi-object semantic discovery.** We report our mACC and mIoU performances averaged over 5 runs with stddev. $K$ refers to the number of classes (including background) and we mark results ($\triangle$) where one outlier run was automatically filtered out.

| Dataset | $K$ | mACC | mIoU |
|---|---|---|---|
| Tetrominoes [2] | 20 | $99.5 \pm 0.2$ | $99.1 \pm 0.4$ |
| Multi-dSprites [7] | 4 | $91.3^{\triangle} \pm 0.9$ | $84.0^{\triangle} \pm 1.4$ |
| CLEVR6 [6, 2] | 7 | $73.9 \pm 2.1$ | $56.3 \pm 2.9$ |

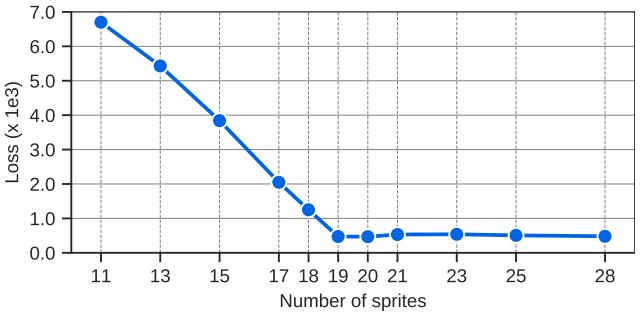


Figure 1: **Effect of $K$.** We report the loss obtained for varying number of sprites on Tetrominoes, where the ground-truth number of different shapes is 19.

## B. Model analysis

### B.1. Effect of $K$

Similar to standard clustering methods, our results are sensitive to the assumed number of sprites $K$. A purely quantitative analysis could be applied to select $K$, *e.g.* in Figure 1 we plot the loss as a function of the number of sprites for Tetrominoes and it is clear an elbow method can be applied to correctly select 19 sprites. Qualitatively, using more sprites than the ground truth number typically yields duplicated sprites which we think is not that harmful. For example, we use an arbitrary number of sprites (40) for the Instagram collections and we have not found the discovered sprites to be very sensitive to this choice.

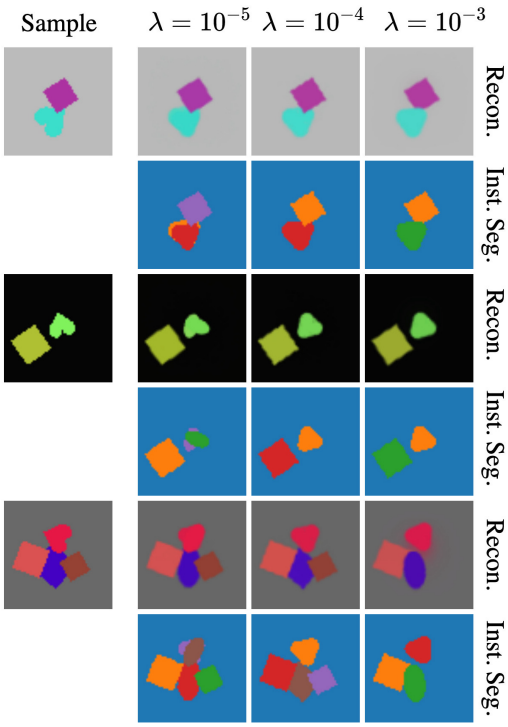


Figure 2: **Effect of $\lambda$.** We show reconstructions and instance segmentations for different values of $\lambda$ on Multi-dSprites.

### B.2. Effect of $\lambda$

The hyperparameter $\lambda$ controls the weight of the regularization term that counts the number of non-empty sprites used. In Figure 2, we show qualitative results obtained for different values of $\lambda$ on Multi-dSprites. When $\lambda$ is zero or small (here $\lambda = 10^{-5}$), the optimization typically falls into bad local minima where multiple layers attempt to reconstruct the same object. Increasing the penalization ($\lambda = 10^{-4}$) prevents this phenomenon by encouraging reconstructions using the minimal number of non-empty sprites. When $\lambda = 10^{-3}$, the penalization is too strong and some objects are typically missed (last example).

### B.3. Computational cost

Training our method on Tetrominoes, Multi-dSprites and CLEVR6 respectively takes approximately 5 hours, 3 days and 3.5 days on a single Nvidia GeForce RTX 2080 Ti GPU. Our approach is quite memory efficient and for example on CLEVR6, we can use a batch size of up to 128 on a single V100 GPU with 16GB of RAM as opposed to 4 in [2] and 64 in [11].

## C. Training details

The full implementation of our approach and all datasets used are available at https://github.com/monniert/dti-sprites.

Table 2: **Transformation sequences used.**

| Dataset | $\mathcal{T}^{\text{lay}}$ | $\mathcal{T}^{\text{spr}}$ | $\mathcal{T}^{\text{bkg}}$ |
|---|---|---|---|
| Tetrominoes [2] | col-pos | - | - |
| Multi-dSprites [7] | col-pos | sim | col |
| CLEVR6 [6, 2] | col-pos | proj | col |
| GTSRB-8 [15] | - | col-proj-tps | col-proj-tps |
| SVHN [14] | - | col-proj-tps | col-proj-tps |
| Weizmann Horse [1] | - | col-proj-tps | col-proj-tps |
| Instagram collections [13] | - | col-proj | col-proj |

### C.1. Architecture

We use the same parameter predictor network architecture for all the experiments. It is composed of a shared ResNet [3] backbone truncated after the average pooling and followed by separate Multi-Layer Perceptrons (MLPs) heads predicting sprite transformation parameters for each layer as well as the occlusion matrix. For the ResNet backbone, we use mini ResNet-32[1] (64 features) for images smaller than $65 \times 65$ and ResNet-18 (512 features) otherwise. When modeling large numbers of objects ($> 3$), we increase the representation size by replacing the global average pooling by adaptive ones yielding $4 \times 4 \times 64$ features for mini ResNet-32 and $2 \times 2 \times 512$ for ResNet-18. Each MLP has the same architecture, with two hidden layers of 128 units.

### C.2. Transformation sequences

Similar to DTI-Clustering [13], we model complex image transformations as a sequence of transformation modules which are successively applied to the sprites. Most of the transformation modules we used are introduced in [13], namely affine, projective and TPS modules modeling spatial transformations and a color transformation module. We augment the collection of modules with two additional spatial transformations implemented with spatial transformers [4]:

- a *positioning module* parametrized by a translation vector and a scale value (3 parameters) and used to model coarse layer-wise object positioning,
- a *similarity module* parametrized by a translation vector, a scale value and a rotation angle (4 parameters).

The transformation sequences used for each dataset are given in Table 2. All transformations for the multi-object benchmarks are selected to mimic the way images were synthetically generated. For real images, we use the col-proj-tps default sequence when the ground truth number of object categories is well defined and the col-proj sequence otherwise. Visualizing sprites and transformations helps understanding the results and adapting the transformations accordingly.

[1] https://github.com/akamaster/pytorch_resnet_cifar10

### C.3. Implementation details

Both sprite parameters and predictors are learned jointly and end-to-end using Adam optimizer [8] with a $10^{-6}$ weight decay on the network parameters. Background, sprite appearances and masks are respectively initialized with averaged images, constant value and Gaussian weights. To constrain sprite parameters in values close to $[0, 1]$, we use a $\mathrm{softclip}$ function implemented as a piecewise linear function yielding identity inside $[0, 1]$ and an affine function with $0.01$ slope outside $[0, 1]$. We experimentally found it tends to work better than (i) a traditional $\mathrm{clip}$ function which blocks gradients and (ii) a sigmoid function which leads to very small gradients. Similar to [13], we adopt a curriculum learning strategy of the transformations by sequentially adding transformation modules during training at a constant learning rate until convergence, then use a multi-step policy by multiplying the learning rate by $0.1$ once convergence has been reached. For the experiments with a single object on top of a background, we use an initial learning rate of $10^{-3}$ and reduce it once. For the multi-object experiments, because spatial transformations are much stronger, we use an initial value of $10^{-4}$ and first train global layer-wise transformations, using frozen sprites during the first epochs (initialized with constant value for appearances and Gaussian weights for masks). Once such transformations are learned, we learn sprite-specific transformations if any and reduce after convergence the learning rate for the network parameters only. Additionally, in a fashion similar to [13], we perform sprite and predictor reassignment when corresponding sprite has been used for reconstruction less than $20/K\%$ of the images layers. We use a batch size of 32 for all experiments, except for GTSRB-8 and SVHN where a batch size of 128 is used.

### C.4. Learning binary masks

There is an ambiguity between learned mask and color values in our raw image formation model. In Figure 3, we show examples of sprites learned following two settings: (i) a raw learning and (ii) a learning where we constrain mask values to be binary. Although learned appearance images $\mathbf{s}^c$ (first row) and masks $\mathbf{s}^\alpha$ (second row) are completely different, applying the masks onto appearances (third row) yields similar images, and thus similar reconstructions of sample images. However, resulting sprites (last row) demonstrate that the spatial extent of objects is not well defined when learning without any constraint.

Since constraining the masks to binary values actually resolves ambiguity and forces clear layer separations, we follow the strategy adopted by Tieleman [16] and SCAE [9] to learn binary values, and propose to inject during training uniform noise $\in [-0.4, 0.4]$ into the masks before applying our $\mathrm{softclip}$. Intuitively, such stochasticity prevents the masks from learning appearance aspects and can only be reduced with values close to $0$ and $1$. We experimentally found this approach tends to work better than (i) explicit regularization penalizing values outside of $\{0, 1\}$ *e.g.* with a $x \rightarrow x^2(1-x)^2$ function and (ii) a varying temperature parameter in a sigmoid function as advocated by Concrete/Gumbel-Softmax distributions [12, 5].

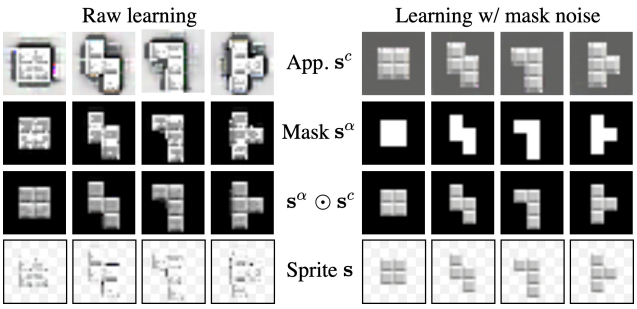


Figure 3: **Learning binary masks.** We compare results on Tetrominoes obtained with (right) and without (left) injecting noise in masks.

We compare our results obtained with and without injecting noise into the masks on Tetrominoes, where shapes have clear appearances. Quantitatively, while our full model reaches almost a perfect score for both ARI-FG and ARI metrics (resp. 99.6% and 99.8%), these performances averaged over 5 runs are respectively 77.8% and 89.1% when noise is not injected into the masks during learning. We show qualitative comparisons in Figure 3. Note that the masks learned with noise injection are binary and sharp, whereas the ones learned without noise contain some appearance patterns.

## D. Additional qualitative results

We provide more qualitative results on the multi-object synthetic benchmarks, namely Tetrominoes (Fig. 4), MultiDSprites (Fig. 5) and CLEVR6 (Fig. 6). For each dataset, we first show the discovered sprites (at the top), with colored borders to identify them in the semantic segmentation results. We then show 10 random qualitative decompositions. From left to right, each row corresponds to: input sample, reconstruction, semantic segmentation where colors refer to the sprite border colors, instance segmentation where colors correspond to different object instances, and full image decomposition layers where the borders are colored with respect to their instance mask color. Note how we manage to successfully separate the object instances as well as identify their categories and spatial extents.

For additional decompositions, we urge the readers to visit imagine.enpc.fr/~monniert/DTI-Sprites/extra_results.

## References

[1] Eran Borenstein and Shimon Ullman. Learning to Segment. In *ECCV*, 2004. 2

Figure 4: **Tetrominoes results.** We show discovered sprites (top) and 10 random decomposition results (bottom).

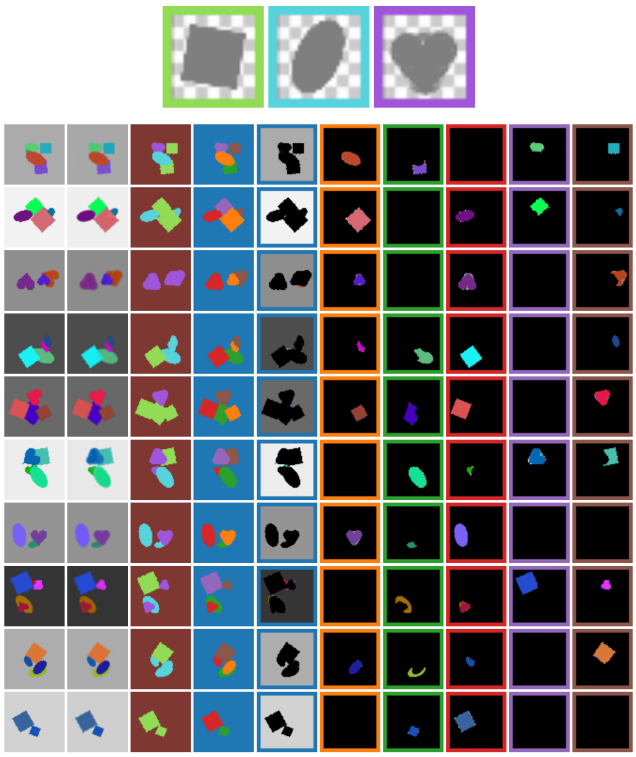

Figure 5: **Multi-dSprites results.** We show discovered sprites (top) and 10 random decomposition results (bottom).

Figure 6: **CLEVR6 results.** We show discovered sprites (top) and 10 random decomposition results (bottom).


[2] Klaus Greff, Raphaël Lopez Kaufman, Rishabh Kabra, Nick Watters, Chris Burgess, Daniel Zoran, Loic Matthey, Matthew Botvinick, and Alexander Lerchner. Multi-Object Representation Learning with Iterative Variational Inference. In *ICML*, 2019. 1, 2

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *CVPR*, 2016. 2

[4] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial Transformer Networks. In *NIPS*, 2015. 2

[5] Eric Jang, Shixiang Gu, and Ben Poole. Categorical Reparameterization with Gumbel-Softmax. In *ICLR*, 2017. 3

[6] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning. In *CVPR*, 2017. 1, 2

[7] Rishabh Kabra, Chris Burgess, Loic Matthey, Raphael Lopez Kaufman, Klaus Greff, Malcolm Reynolds, and Alexander Lerchner. Multi-object datasets.

https://github.com/deepmind/multi_object_datasets/, 2019. 1, 2

[8] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *ICLR*, 2015. 3

[9] Adam Kosiorek, Sara Sabour, Yee Whye Teh, and Geoffrey E Hinton. Stacked Capsule Autoencoders. In *NeurIPS*, 2019. 3

[10] H. W. Kuhn and Bryn Yaw. The Hungarian method for the assignment problem. *Naval Research Logistic Quarterly*, 1955. 1

[11] Francesco Locatello, Dirk Weissenborn, Thomas Unterthiner, Aravindh Mahendran, Georg Heigold, Jakob Uszkoreit, Alexey Dosovitskiy, and Thomas Kipf. Object-Centric Learning with Slot Attention. In *NeurIPS*, 2020. 2

[12] Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables. In *ICLR*, 2017. 3

[13] Tom Monnier, Thibault Groueix, and Mathieu Aubry. Deep Transformation-Invariant Clustering. In *NeurIPS*, 2020. 2, 3

[14] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading Digits in Natural Images with Unsupervised Feature Learning. In *NIPS Workshop*, 2011. 2

[15] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*, 2012. 2

[16] Tijmen Tieleman. *Optimizing Neural Networks That Generate Images*. PhD Thesis, University of Toronto, 2014. 3