Learned Equivariant Rendering without Transformation Supervision

Cinjon Resnick* NYU
Or Litany NVidia
Hugo Larochelle Google
Joan Bruna NYU
Kyunghyun Cho NYU

Abstract

We propose a self-supervised framework to learn scene representations from video that are automatically delineated into objects and background. Our method relies on moving objects being equivariant with respect to their transformation across frames and the background being constant. After training, we can manipulate and render the scenes in real time to create unseen combinations of objects, transformations, and backgrounds. We show results on moving MNIST with backgrounds.

1 Introduction

Learning manipulable representations of scenes is a challenging task. Ideally, we would give our model an image of a scene and receive an inverse rendering of coherent objects along with a static background. This is infeasible without an inductive bias because the problem is ill-posed. For example, distinguishing a previously unseen object from the background is inherently ambiguous. Consequently, prior works introduce mechanisms to help the model at training or inference time. We do similarly and choose a mechanism that intuitively follows from learning from video.

Assume an input video of a dynamic scene captured by a static camera. The static background is the implicit objects that are constant across the frames[2]. In contrast, the foreground is the implicit objects that are equivariant with respect to a family of transformations containing more than just the identity. We use this difference to infer a scene representation that captures separately the background and the moving object. Like Dupont et al. [1], we limit the family of transformations to be affine. Building on their work, we attain two novel results:

1. **Learn the transformation**: We learn the tranformation from nearby video frames and do not require it as input during training. This has further benefits at inference time.
2. **Distinguish objects and background**: By encoding the background as the constant objects and the manipulable character as the equivariant object, we yield orthogonal encoders for the character and the background and can consequently manipulate them independently.

Our model is trained in a self-supervised fashion with only rendered pairs (x_p^i, x_q^i) of nearby frames from video sequence i. We impose no other constraint and can render new scenes combining objects and backgrounds in real time with additional (potentially unrelated) frames x^j and x^k. We show strong results in Sec 4 on a 2D task involving moving MNIST[3] digits on static backgrounds where we demonstrate the following manipulations:

- Render the object in x_1^i but with the background from x_1^j.
- Render the object in x_1^i using the transformation seen from $x_p^j \to x_q^j$.

*Correspondence to cinjon@nyu.edu.

[2]Note that with this definition of the background, occlusions are just occasions when the implicit object is not visibly painted in the scene. Practically, this can be handled with alpha transparency.

Workshop on Differentiable Vision, Graphics, and Physics in Machine Learning at NeurIPS 2020.

- Combine the above two manipulations to render the object in x_1^i on the background from x_1^j using the transformation exhibited by the change seen from $x_p^k \rightarrow x_q^k$.

2 Background

Dupont et al. [1] denoted an image $x \in X = \mathbb{R}^{c \times h \times w}$ along with a scene representation $z \in Z$, a rendering function $g : Z \rightarrow X$ mapping scenes to images, and an inverse renderer $f : X \rightarrow Z$ mapping images to scene representations. It is helpful to consider x as a 2D rendering of an implicit object o from a specific camera viewpoint. Then, with respect to transformation T, an equivariant scene representation z satisfies the following relation:

$$T^X g(z) = g(T^Z z) \tag{1}$$

In other words, transforming a rendering in image space is equivalent to first transforming the scene in feature space and then rendering the result. Here, x is a 2D image rendering, z is that rendering's scene representation, and T^Z is an affine transformation. We then learn neural networks f and g s.t.:

$$x_2 = T^X x_1 = T^X g(z_1) = g(T^Z z_1) = g(T^Z f(x_1)) \tag{2}$$

In [1], they require triples (x_1, x_2, θ). The pair x_1 and x_2 are renderings of the same object o, and θ is the rotation transforming o from its appearance in x_1 to its appearance in x_2. They use the same θ for T^Z, which means that the 3D representation $z_1 = f(x_1)$ is rotated by θ. This yields $\tilde{z}_1 = R_\theta^Z f(x_1)$ and $\tilde{z}_2 = (R_\theta^Z)^{-1} f(x_2)$, with which they train g and f to minimize reconstruction loss (Eqn 3):

$$\mathbb{L}_{\text{render}} = ||x_2 - g(\tilde{z}_1)|| + ||x_1 - g(\tilde{z}_2)|| \tag{3}$$

Figure 1: **Inference demonstration**: The first three rows are ground truth from the test set. The fourth row is the object from the third row, the transformations in the second row, and the background from the first row.

After training, they infer new renderings by first inverse rendering $z = f(x)$, then applying rotations T^Z upon z, and finally rendering images $\hat{x} = g(T^Z z)$. This is notable because it means we can operate entirely in feature space. As we show in Sec 3, this lets us manipulate the output rendering in ways that are very difficult to perform in image space.

3 Method

Our motivation is in asking if we can we use the smoothly-changing nature of video to learn the transformations between frames. Like [1], we assume that the change between frame F_t and F_{t+1} can be modeled with affine transformations. However, while they use one (rotation) transformation, we assume an arbitrary affine transformation on the character object plus an invariant transformation on a static background (in-painting as needed). Additionally, we remove the θ requirement at both training and test by learning the transformation T^Z from data.

Learning the transformation That video changes smoothly lets us advance past affine transformations parametrically defined with a θ angle of rotation and instead learn transformations based on frame changes. One advantage is that the model is now agnostic to which affine transformations the data exhibits. Another is that at inference time, the model is agnostic to whether the frame to be transformed is input to the transformation function.

Distinguishing objects and background The renderer g is equivariant to moving objects. A static background however will be constant across a video. We take advantage of this to learn g along with functions f_o and f_b corresponding to the moving object and the background such that, at inference time, we can mix and match objects and backgrounds that we have previously never seen together.

Setup Building on Sec 2, we define f_b and f_o respectively for the encoding of the background and the object. Following Eqn 2, we then learn neural networks f_o, f_b, g, and T^Z such that:

$$x_2 = g\left(T^Z(f_o(x_1), f_o(x_2)) \circ f_o(x_1) + f_b(x_1)\right) \tag{4}$$

During training, we require only the pair (x_1, x_2). We find that two more constraints help. The first is that the object encoder is equivariant with respect to the transformation. The second is that the background encoder is constant. These are described below where we optimize $\mathbb{L}_{\text{total}}$ with scalar coefficients $\alpha_{\text{equiv}}, \alpha_{\text{inv}}$:

$$\mathbb{L}_{\text{scene}} = ||g\left(T^Z(f_o(x_1), f_o(x_2)) \circ f_o(x_1) + f_b(x_1)\right) - x_2||_2 \tag{5}$$

$$\mathbb{L}_{\text{equiv}} = ||T^Z(f_o(x_1), f_o(x_2)) \circ f_o(x_1) - f_o(x_2)||_2 \tag{6}$$

$$\mathbb{L}_{\text{inv}} = ||f_b(x_1) - f_b(x_2)||_2 \tag{7}$$

$$\mathbb{L}_{\text{total}} = \mathbb{L}_{\text{scene}} + \alpha_{\text{equiv}}\mathbb{L}_{\text{equiv}} + \alpha_{\text{inv}}\mathbb{L}_{\text{inv}} \tag{8}$$

With this setup, both T^Z and f_o learn to handle every object similarly. This is important because it means that at inference time we can render novel scenes given a pair of nearby frames (x_1, x_2) in a video. Denoting $h(x_1, x_2, x_3, x_4) = g\left(T^Z(f_o(x_1), f_o(x_2)) \circ f_o(x_3) + f_b(x_4)\right)$, the novel renderings described in Sec 1 and shown in Sec 4 are:

- $\mathbf{h(x_1^i, x_2^i, x_1^i, x_1^j)}$: Render the object in x_1^i as it is in x_2^i but with the background from x_1^j.
- $\mathbf{h(x_1^i, x_2^i, x_1^j, x_1^j)}$: Render the object in x_1^j using the transformation exhibited by the change in the object from $x_1^i \to x_2^i$.
- $\mathbf{h(x_1^i, x_2^i, x_1^j, x_1^k)}$: Combine the above two to render the object in x_1^j on the background from x_1^k using the transformation exhibited by the change in the object from $x_1^i \to x_2^i$.

4 Experiments

We show experiments on a dataset built on MNIST. This test-bed is suitable for demonstrating that our method is capable of both learning the transformations and separating objects from the background.

Dataset We generate videos, each of length $M = 5$, of MNIST digits (objects) moving on a static background. The digits and background have dimensions $(28, 28)$ and $(64, 64)$ respectively. At each training step, we select N digits in the train split of MNIST, as well as a background from the set of pre-generated training backgrounds (see below). We then place these digits at some random initial position. For each digit, and for each of $M - 1$ times, we choose randomly between either rotation or translation. If we choose translation, then we translate the object independently in each of the x and y directions by some random amount in $[-10, -8, \ldots, 8, 10] \setminus 0$. If we choose rotation, then we rotate the object by some random amount in $[-15, -12, \ldots, 12, 15] \setminus 0$. In both cases, if character leaves the boundaries of the image, then we redo the transformation selection. Otherwise, that transformation is applied cumulatively to yield the next object position.

At this point, we have MNIST images on blank canvases. We overlay them on the chosen background to produce a sequence of images where the change in each object from frames $F_T \to F_{T+1}$ is small and affine for the object and constant for the background. Afterwards, we randomly choose two indices i, j and use (x_i, x_j) as the training pair. See Figure 1 for example sequences.

Backgrounds We create 64 randomly generated backgrounds for each of train and test. For each background, we select a color from the Matplotlib CSS4 colors list. We then place five diamonds on the background, each with a different random color, along with an independent and randomly chosen center and radius. The radius is uniformly chosen from between seven and ten, inclusive.

Model We use neural networks for f_o, f_b, g, and T^Z. While distinct, both f_o and f_b share the same architecture details. The renderer g is a transposition of that architecture, albeit without being residual. The transformation T^Z has three important aspects. First, it is input-order dependent. Second, it uses PyTorch's[5] affine_grid and grid_sample functions to transform the scene similarly to how Spatial Transformers[2] operate. The third is that it is initialized at identity.

Qualitative Results Our model learns to render new scenes using objects from the test set of MNIST as well as backgrounds it has never seen before. All shown sequences are on unseen backgrounds with unseen MNIST digits where there at least two transformations of each type (rotation and translation) and the transformations were cumulatively large over the sequence. We did not need to cherry-pick any of the results.

Figure 2 concisely demonstrates the manipulations from Sec 3. The first three rows are ground truth from the test set. The fourth row is replacing the background in the first row with that of the second row. The fifth row is applying the transformations in the third row to the first row's character and background. And the sixth row is both manipulations simultaneously.

In particular, the final row is rendered by encoding the character from the first row, the background from the second row, and using the transformations exhibited in the third row. With x_j^i as the jth frame from the ith sequence:

$$x_k^6 = g\left(T^Z(f_o(x_1^3), f_o(x_k^3)) \circ f_o(x_k^1) + f_b(x_k^2)\right)$$

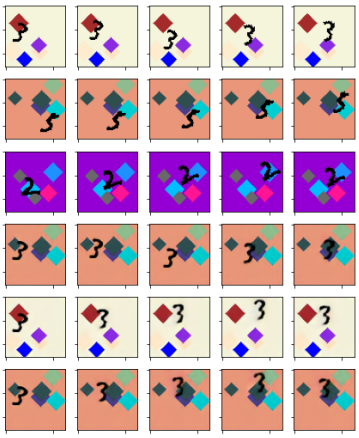

Figure 2: Section 3 manipulations. With counterclockwise direction and bottom left origin, the transformations in the third row are rotate(15), rotate(9), translate(10, 4), translate(-8, -6).

Quantitative Results We tested reconstruction results by evaluating the per pixel float MSE over the MNIST test set. For each example, we randomly chose two pairs of (background, digit) and made corresponding videos $(x_1^1, \ldots, x_5^1)$ and $(x_1^2, \ldots, x_5^2)$. We then indexed into the same random position in both sequences to get frame pairs (x_i^1, x_j^1), (x_i^2, x_j^2).

To get the MSE of transformation manipulations, we render the object and background from x_i^2 but transformed like in $x_i^1 \rightarrow x_j^1$. We compare this ground truth to $\hat{x}_j^2 = g(T^Z(f_o(x_i^1), f_o(x_j^1)) \circ f_o(x_i^2) + f_b(x_i^2)$. To get the MSE of background manipulations, we render the object in x_j^1 on the background from x_j^2 as ground truth and compare it to $\hat{x}_j^1 = g(T^Z(f_o(x_i^1), f_o(x_j^1)) \circ f_o(x_i^1) + f_b(x_i^2)$.

Fig 3 shows a boxplot of these results along with two baselines: *Video frames* is the MSE of two random frames from the same video; *No object* is the MSE of a full frame against only the background from that frame. Given that MSE is a measure of reconstruction quality with lower values being better, we expect them to serve as upper bounds. *Video frames* is the upper bound when reconstruction gets the object but places it incorrectly. *No object* is the upper bound when reconstruction fails to include the object. On this measure, we see that the background manipulation is much better than the baselines, but we cannot say with certitude that the transform manipulation is better as it is within confidence interval of *Video Frames* and its box plot overlaps with both baselines.

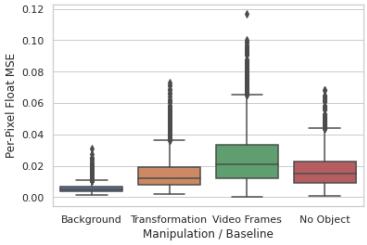

Figure 3: Per-pixel MSE over 10,000 test examples. The transform and background manipulations use our learned functions; Video frames is MSE of a frame against a random (non-identical) frame from the same video; No object is MSE of the background versus the full frame of background and object.

5 Related Work

Besides [1], two other related works are Worrall et al. [7], Olszewski et al. [4]. They also rely on equivariance to learn representations capable of manipulating scenes. However, they do not delineate objects and backgrounds, nor do they learn the T^Z from data. Dupont et al. [1] assumes that T^Z is given during training; In Worrall et al. [7], T^Z is a block diagonal composition of (given) domain-specific transformations. Olszewski et al. [4] uses a user-provided transformation. That we learn it from data lets us work with datasets where we do not have ground truth.

Reed et al. [6] is also related. They apply transformations to frames to yield an analogous frame. However, they assume that the applying operation is addition rather than spatial transform, and they require an additional frame as input to T^Z at inference (3) and two additional during training (4).

6 Conclusion

In this work, we have presented a framework for learning an equivariant renderer capable of delineating objects and the background such that it can manipulate each independently. Further, our framework only requires self-supervised video sequences and does not need labels.

Our assumption that the transformation between frames is affine does not hold in general; a contrary example is videos with nonlinear lighting effects. While there are real applications where it does occur such as stop-motion animation, relaxing this assumption is an area we are actively considering. A more pressing direction though is increasing the complexity of the datasets on which we experiment. We leave that extension to future work.

References

[1] Emilien Dupont, Miguel Angel Bautista, Alex Colburn, Aditya Sankar, Carlos Guestrin, Josh Susskind, and Qi Shan. Equivariant neural rendering, 2020.

[2] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial transformer networks. *CoRR*, abs/1506.02025, 2015. URL http://arxiv.org/abs/1506.02025.

[3] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010. URL http://yann.lecun.com/exdb/mnist/.

[4] Kyle Olszewski, Sergey Tulyakov, Oliver J. Woodford, Hao Li, and Linjie Luo. Transformable bottleneck networks. *CoRR*, abs/1904.06458, 2019. URL http://arxiv.org/abs/1904.06458.

[5] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[6] Scott E Reed, Yi Zhang, Yuting Zhang, and Honglak Lee. Deep visual analogy-making. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 1252–1260. Curran Associates, Inc., 2015. URL http://papers.nips.cc/paper/5845-deep-visual-analogy-making.pdf.

[7] Daniel E. Worrall, Stephan J. Garbin, Daniyar Turmukhambetov, and Gabriel J. Brostow. Interpretable transformations with encoder-decoder networks. *CoRR*, abs/1710.07307, 2017. URL http://arxiv.org/abs/1710.07307.

Appendix

Further examples Here, we show more examples of our model manipulating sequences.

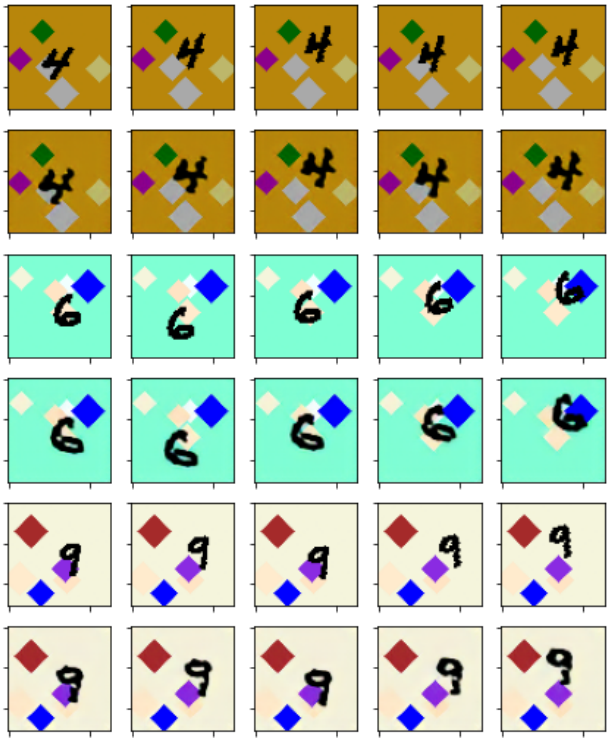

Figure 4: **Reconstructions**: The first, third, and fifth rows are original sequences. The second, fourth, and sixth rows are reconstructions of the prior row where T^Z is fixed as the same transformation as T^X. Note that in these scenarios, the results are not as strong as when T^Z is a learned function.

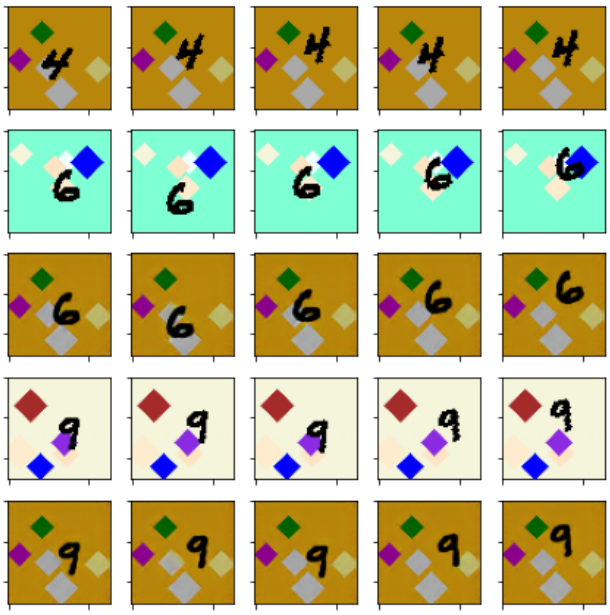

Figure 5: **Backgrounds**: The first, second, and fourth rows are originals. The third and fifth rows are the prior row but with the background changed to that of the first sequence.

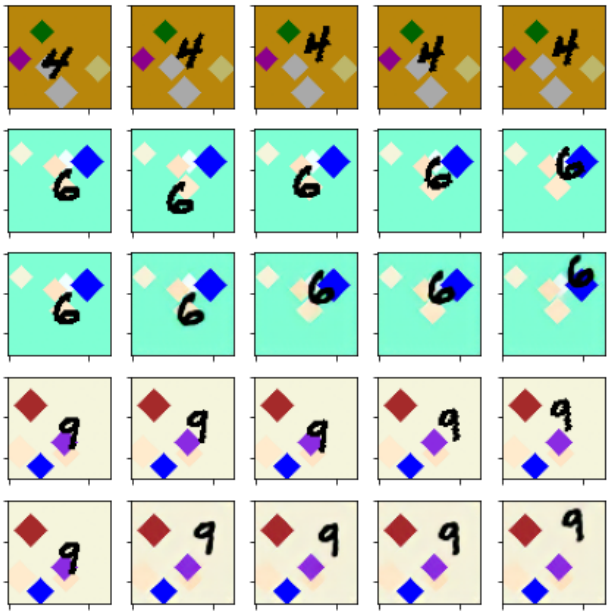

Figure 6: **Transformations**: The first, second, and fourth rows are originals. The third and fifth rows are the prior row but the transformation is a function of the first row.

Analyzing T^Z We compare the learned T^Z to the ground truth object transformation. Each column in 1 shows a 2×3 matrix representing the independent statistics of each entry of the transformation in question. These statistics are attained over 40,000 unique frame pairs. For example, the first column shows the mean of each entry in the transformation matrix.

Transformation	Mean	Max	Min
Ground Truth	$\begin{pmatrix} 0.990 & -0.003 & 0.558 \\ 0.003 & 0.990 & 0.424 \end{pmatrix}$	$\begin{pmatrix} 1.000 & 0.500 & 26.74 \\ 0.588 & 1.000 & 30.00 \end{pmatrix}$	$\begin{pmatrix} 0.809 & -0.588 & -28.00 \\ -0.500 & 0.809 & -26.00 \end{pmatrix}$
Learned T^Z	$\begin{pmatrix} 1.197 & 0.050 & -0.026 \\ 0.178 & 1.109 & -0.023 \end{pmatrix}$	$\begin{pmatrix} 1.061 & -0.324 & -1.066 \\ -0.063 & 1.026 & -1.157 \end{pmatrix}$	$\begin{pmatrix} 1.268 & 0.362 & 0.936 \\ 0.314 & 1.262 & 0.889 \end{pmatrix}$

Table 1: Independent statistics of the transformations.

Example background The diamond colors and radii are randomly chosen and distinct.

Figure 7: Example background.

PSNR of Baselines and Manipulations PSNR over 10,000 test examples. The transform and background manipulations are done with our learned functions. Video frames is the PSNR of a frame against a random (non-identical) frame from the same video. No object is the PSNR of the background versus the full frame of background plus object.

Type	PSNR 95% CI
Background manipulation	$22.920 \pm .033$
Transform manipulation	$18.912 \pm .051$
Baseline: Video frames	18.993 ± 0.319
Baseline: No object	$18.609 \pm .060$

Table 2: PSNR.