Supplementary Material

1. Data and Code Availability

All code, including code for all simulations and code used to generate the VAEC dataset, is available on GitHub. The dataset can be downloaded from DataSpace.

2. Architecture and Training Procedure for Visual Analogy Dataset

For the visual analogy dataset from Hill et al. (2019), we used an architecture and training procedure modeled as closely as possible on those used in the original paper, so as to facilitate a direct comparison between their model and ours (essentially the original model, with the addition of TCN).

The model architecture consisted of a feedforward encoder, a recurrent network, and a linear scoring layer. After resizing the images from 160×160 to 80×80, they were passed to the feedforward encoder, which consisted of 4 convolutional layers, each with 32 kernels of size 3×3, and a stride of 2, resulting in a feature map of size $5 \times 5 \times 32$. This feature map was then flattened to produce the image embedding. We then applied either TCN or batch normalization to the embeddings, and passed the embeddings for each candidate analogy (consisting of 3 source images, 2 target images, and a candidate answer) to the recurrent network. The recurrent network was a simple RNN, with 64 hidden units. The final hidden state of the RNN for each candidate analogy was then passed through a linear layer to generate a score for the analogy.

The scores for all candidate analogies were passed through a softmax layer, and the network was trained to maximize the probability of the correct answer, using a cross entropy loss. We also employ the training method ('Learning Analogies by Contrasting') advocated by Hill et al. (2019) for all simulations with this dataset. Each network was trained with a batch size of 32 over 3 full epochs with the entire training set of $600,000$ analogy problems (we found that all networks reached a clear asymptote by this point in training). We used the ADAM optimizer (Kingma & Ba, 2014) with a learning rate of $1e^{-}4$. For the feedforward encoder, all weights were initialized using Kaiming normal initialization (He et al., 2015). For the recurrent network and linear scoring layer, weights were initialized using Xavier normal initialization (Glorot & Bengio, 2010). All biases were initialized to zero. All simulations for this dataset were performed using PyTorch (Paszke et al., 2017).

3. Training Time

In the work presented here, we have focused on the extent to which our proposed approach, TCN, improved the ability of neural networks to extrapolate. However, a more common reason for employing normalization techniques in deep learning is to accelerate training (Ioffe & Szegedy, 2015; Bjorck et al., 2018). We therefore also investigated whether TCN can provide a similar benefit in terms of training time.

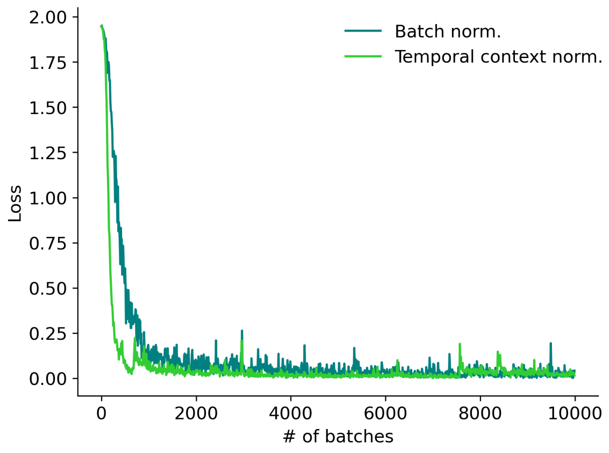

Figure S1. Training loss time courses for models trained with temporal context normalization (TCN) or batch normalization. Each line reflects an average of 8 runs.

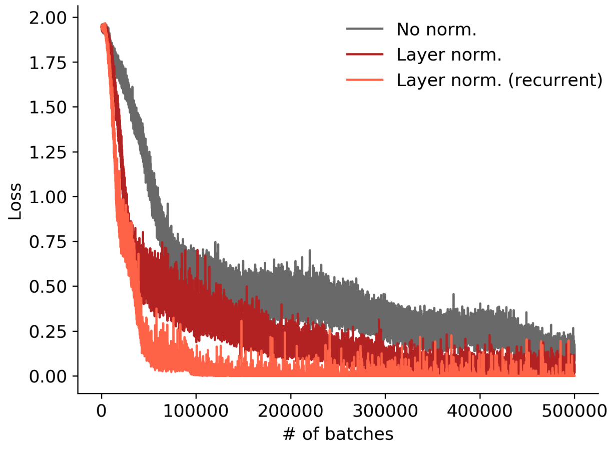

Figure S2. Training loss time courses for models trained with layer or no normalization. Each line reflects an average of 8 runs.

Table S1. Analysis of the TCN model on the Translation Extrapolation regime of the VAEC dataset, separated by individual dimensions of variation. Results show accuracy in each region, including the training region (Region 1), averaged over 8 trained networks ($\pm$ the standard error of the mean).

	Region 1 (training)	Region 2	Region 3	Region 4	Region 5	Region 6
Size	97.5 ± 1.9	81.1 ± 5.3	79.8 ± 8.4	79.4 ± 8.7	80.7 ± 10.0	77.0 ± 11.0
Brightness	99.0 ± 0.6	97.4 ± 1.6	97.1 ± 1.2	93.4 ± 2.1	90.0 ± 4.6	87.0 ± 8.0
Location (X)	99.9 ± 0.1	55.4 ± 12.2	43.9 ± 9.6	47.5 ± 9.6	50.7 ± 10.2	37.3 ± 7.2
Location (Y)	100.0 ± 0.0	74.3 ± 9.7	73.2 ± 11.7	64.8 ± 11.3	61.4 ± 10.5	56.5 ± 9.9

Figures S1 and S2 compare training loss time courses on the Translation Extrapolation regime of the VAEC dataset. Each figure shows training loss averaged across all 8 runs for a particular technique. We find that models trained with both TCN and batch normalization (Figure S1) converge considerably faster than models trained with layer normalization or no normalization at all (Figure S2; note the difference in scale of the X axes). We conclude from these results that, in addition to providing a benefit in terms of extrapolation, TCN also provides a training speedup comparable to that enabled by batch normalization.

4. Analysis of Performance Separated by Individual Dimensions of Variation

The VAEC dataset contains analogy problems in which objects vary in terms of their size, brightness, or location. To determine whether some of these dimensions were more challenging than others, we analyzed performance in each dimension separately. Table S1 shows the results of this analysis. Extrapolation was strongest for the brightness dimension. This may be because this dimension is already represented in the input in a manner that captures the linear structure of the dimension (as a real number). By contrast, our model struggled most with the location dimension. This may be due in part to the convolutional layers in the feedforward encoder, which discard information about location by design. This suggests that one path for improving extrapolation beyond the present results might be to augment the encoder with a scheme for representing spatial location, such as that advocated by Liu et al. (2018).

5. Learned Representations

To better understand how TCN facilitated extrapolation, we also analyzed the latent representations learned by the networks. In our network architecture, each image in a sequence, $\mathbf{x}_t$, was processed by a feedforward encoder, yielding a low-dimensional embedding, $\mathbf{z}_t$, that was then passed to a recurrent network. We focused our analysis on these learned low-dimensional embeddings.

We performed principal component analysis (PCA) on the learned image embeddings for networks trained on the visual analogy task. PCA was performed separately for the images in each of the training and test regions. For networks trained with TCN, we also applied TCN to the embeddings (using the embeddings for the other images in an analogy problem as context) before performing PCA. For networks trained with batch normalization, we applied batch normalization to the embeddings before performing PCA.

We found that a large amount of variance was captured by the first principal component (PC) for all networks. The first PC accounted for an average of 80% of the variance for networks trained with TCN, an average of 62% of the variance for networks trained with batch normalization, and an average of 90% of the variance for networks trained with no normalization. Though this may seem surprising, given that the objects in the visual analogy task varied along four dimensions, it can be explained by the fact that, for any given analogy problem, objects only varied along one dimension. Thus, the task can be effectively solved by mapping the objects in an analogy problem to a single dimension, treating the values in irrelevant dimensions as constants.

For each image embedding, we plotted the value of the first PC against the values of the four underlying dimensions of variation. For networks trained with either TCN or batch normalization, we restricted each plot to embeddings from analogy problems in which objects varied along the corresponding dimension (e.g. when plotting the first PC against the size of the object, we restricted the plot to embeddings from analogies in which the size of the object varied). For networks trained without normalization, the learned embeddings did not depend in any way on the other objects in an analogy problem, so this restriction was not applied when plotting the embeddings for networks trained without normalization.

Additionally, for networks trained with TCN or batch normalization, we applied the corresponding form of normalization (either TCN or batch normalization) to the underlying values themselves. We did this because we were primarily interested in the way in which the learned embeddings captured the relative values of the objects, rather than their absolute values. For networks trained without normalization, we used the absolute values, since there was no corresponding normalization procedure to perform.

The results of this analysis are shown in Figures S3 - S8. The results shown reflect the learned embeddings from the best performing network (as determined by average accuracy in all test regions) with each technique, but the results were qualitatively similar in all networks. In the training region (Figure S3), for networks trained with TCN, the values along the first PC of the learned embeddings closely tracked a linear function of the underlying values, mirroring the linear structure of the dimensions themselves. By contrast, for networks trained with batch normalization, the relationship between the first PC and underlying values only weakly matched a linear fit. For networks trained without normalization, there was a non-monotonic relationship between the first PC and the underlying values, suggesting that other principal components were likely necessary to accurately represent the values of the objects.

Moreover, for networks trained with TCN, the representations learned in the training region showed a considerable degree of similarity to the representations in the test regions (Figures S4 - S8; note that the orientation of the linear fits in each region is arbitrary, given that the PCA was performed separately in each region). By contrast, for networks trained with batch normalization or no normalization, there is a greater degree of variability between the representations in each of the training and test regions. For instance, the non-monotonic function observed in the training region for networks trained without normalization is completely absent in the test regions.

In summary, networks trained with TCN learned a representation that more closely matched the linear structure of the underlying dimensions, and that was better preserved across the test regions, likely resulting in the improved capacity for extrapolation observed in these networks.

References

Bjorck, N., Gomes, C. P., Selman, B., and Weinberger, K. Q. Understanding batch normalization. In *Advances in Neural Information Processing Systems*, pp. 7694–7705, 2018.

Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, 2010.

He, K., Zhang, X., Ren, S., and Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

Hill, F., Santoro, A., Barrett, D. G., Morcos, A. S., and Lillicrap, T. Learning to make analogies by contrasting abstract relational structure. *arXiv preprint arXiv:1902.00120*, 2019.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Liu, R., Lehman, J., Molino, P., Such, F. P., Frank, E., Sergeev, A., and Yosinski, J. An intriguing failing of convolutional neural networks and the coordconv solution. In *Advances in Neural Information Processing Systems*, pp. 9605–9616, 2018.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. Automatic differentiation in pytorch. 2017.

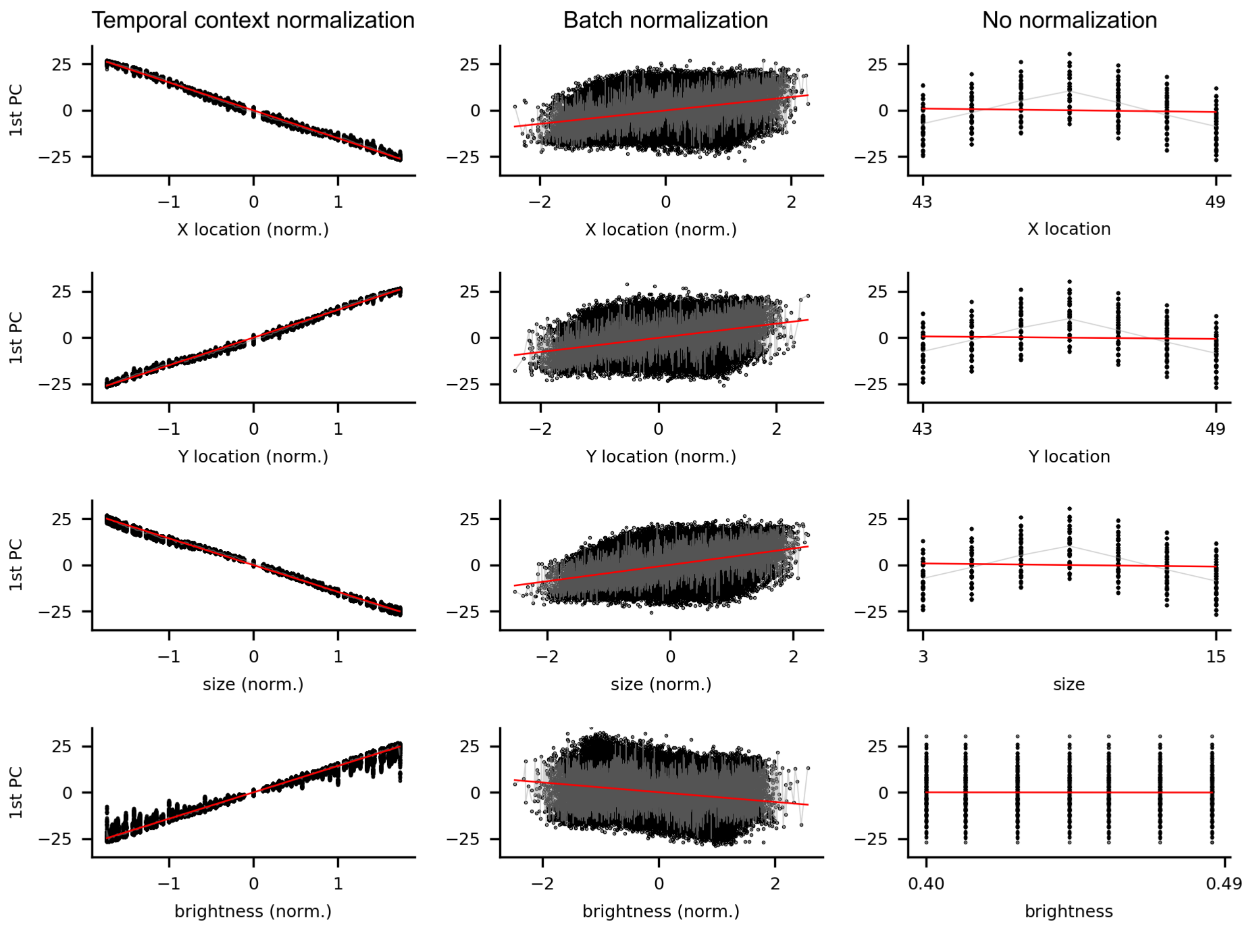

Figure S3. PCA results in Region 1 (training). Y axis represents value along 1st PC. X axis represents value along underlying dimension of variation (location, size, or brightness). Gray line represents the average of all points for each unique value along the X axis. Red line represents a linear fit.

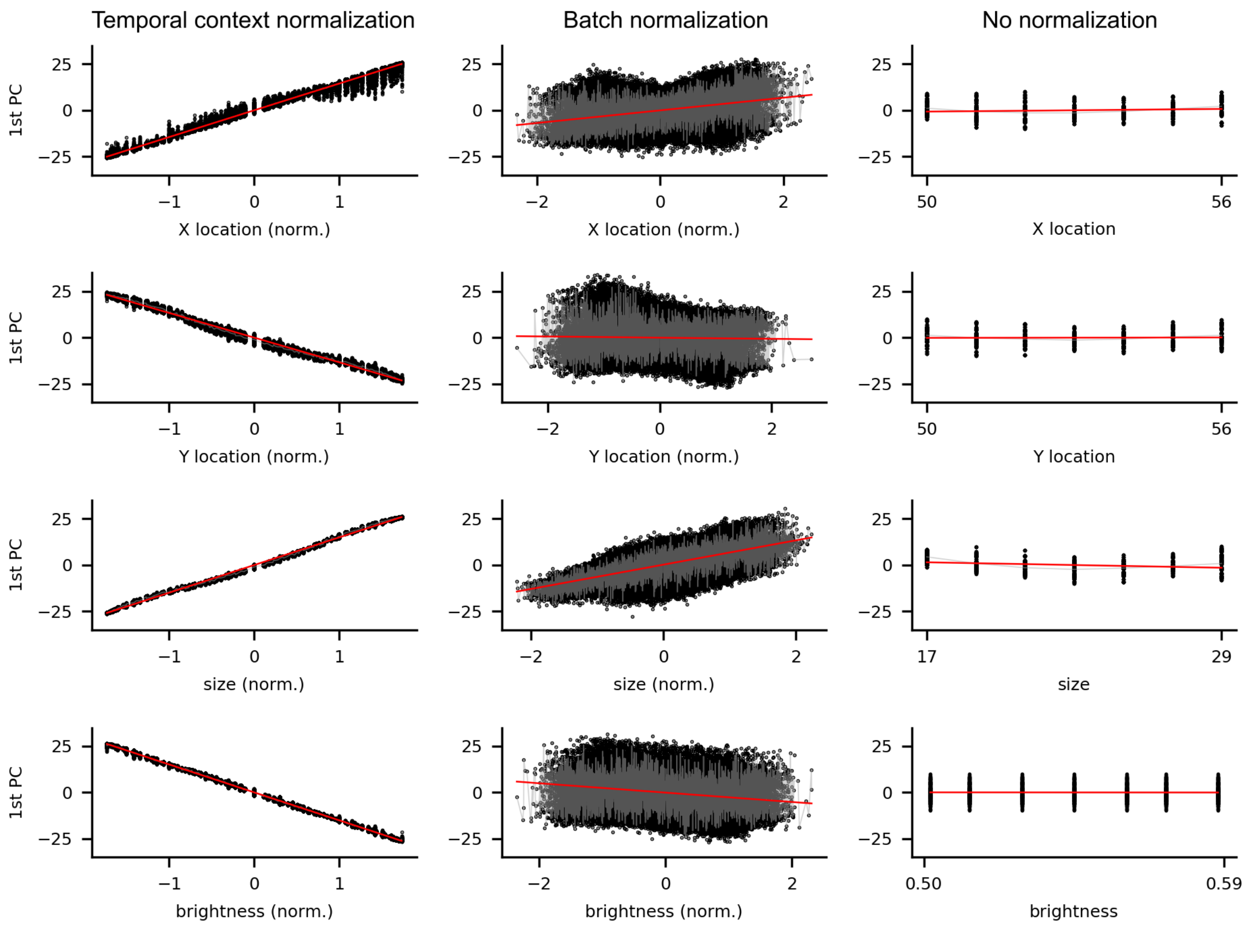

Figure S4. PCA results in Region 2.

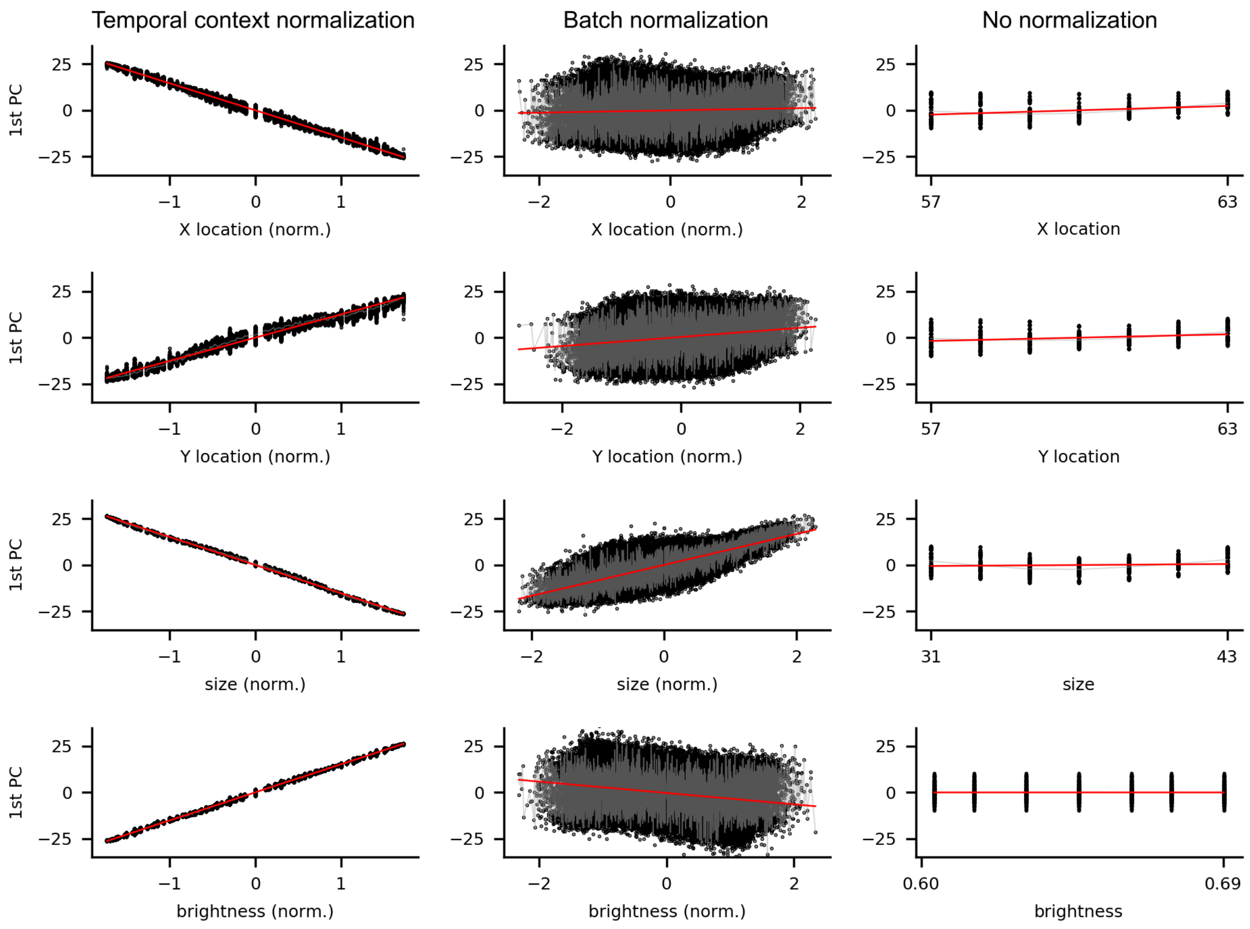

Figure S5. PCA results in Region 3.

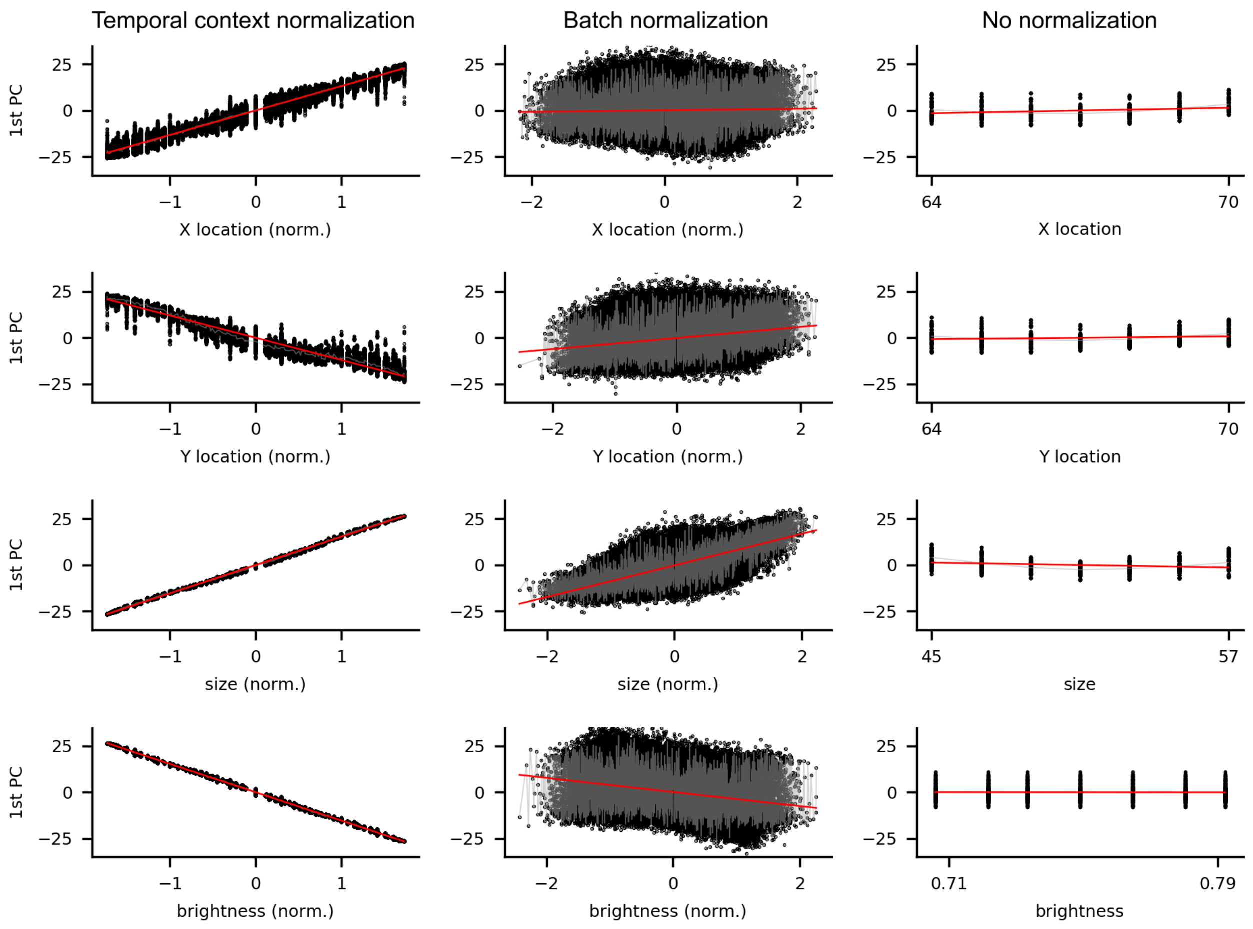

Figure S6. PCA results in Region 4.

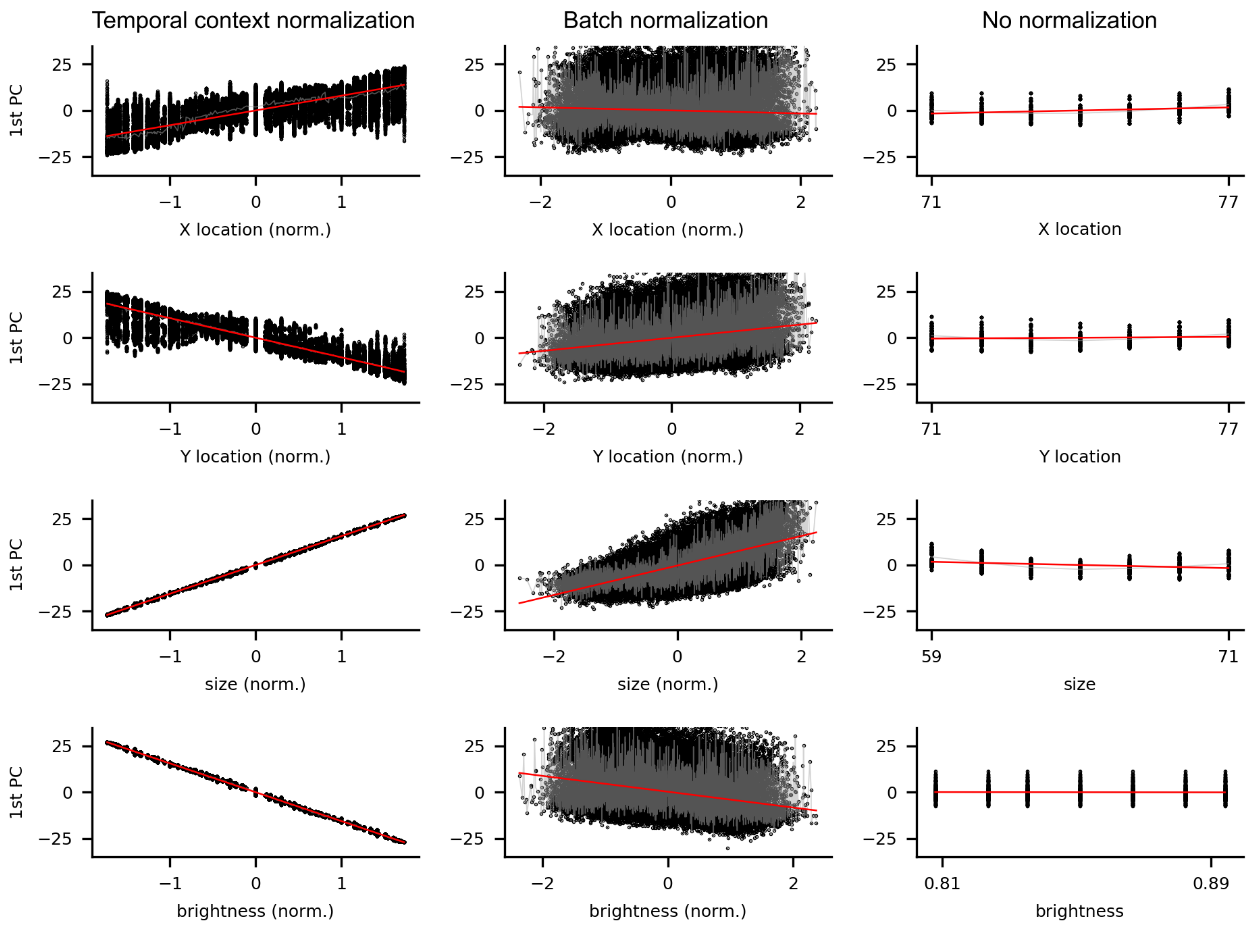

Figure S7. PCA results in Region 5.

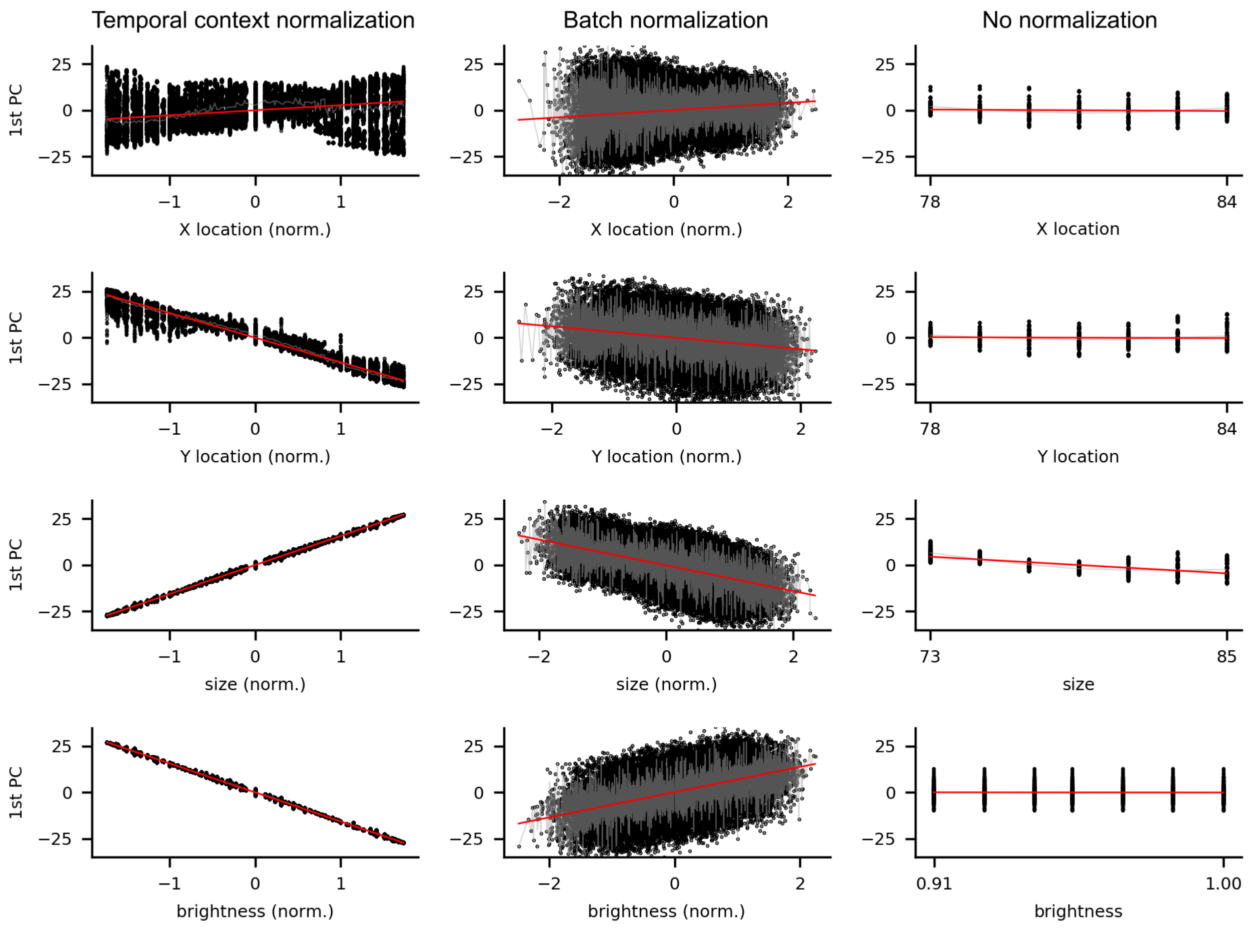

Figure S8. PCA results in Region 6.