Batch Normalization using an SVD Layer

Abdulmohsen Mohammedsaleh
KAUST Academy
abdulmohsen1284@gmail.com

Majed Alsulami
KAUST Academy
majedsaadalsulami@gmail.com

Yousef Alogiley
KAUST Academy
yousef.alogiley@gmail.com

Mentor: **Jad Mounayer** jad.mounayer@outlook.com
KAUST Academy Summer Program 2026, University of Oxford stage • Code: github.com/Yousef10p/RR-Layer

Abstract

Batch Normalization (BN) is the standard way to control the statistics of activations inside a neural network. Rank Reduction Autoencoders showed that the truncated Singular Value Decomposition (SVD) can regularize an autoencoder without any extra loss term, but the effect of the SVD by itself, separated from the autoencoder, has not been studied. We isolate it into a standalone **SVD layer**: a drop-in PyTorch module that rebuilds each mini-batch from its top-k singular components during training and projects onto a frozen inference basis at test time. We evaluate the layer as a direct BN replacement on a controlled point-source localization task (regressing the center of a Gaussian source from a 64×64 image), comparing Plain, BN, and SVD-layer networks trained from identical initial weights over up to 32 seeds, sweeping rank, noise, dataset size, and background type. The SVD layer beat BN in every comparison we ran (clean data, noisy data, small training sets, and real image backgrounds), by factors of 1.5–15×. Under heavy input noise it also beat the unregularized network at every training-set size we tested (2–3× lower test MSE), with the smallest seed variance and a near-zero generalization gap. On real, structured backgrounds the plain network remained best, which localizes the layer's weakness: keeping the top singular components also keeps low-rank background clutter. We also show that early training blow-ups were caused by a hand-written SVD gradient, not by the layer itself; with PyTorch's native SVD autograd the layer is stable at every position in the network. We release the layer as a small PyTorch module that can be dropped into any `nn.Module`.

1 Introduction

Batch Normalization [2] rescales every feature of a mini-batch to mean 0 and variance 1, followed by a learned affine map:

$$y = \gamma \cdot (x - \mu_B) / \sigma_B + \beta$$

This simple step lets networks train faster, reach higher accuracy, and keeps activation distributions stable through training, which is why BN is a standard layer in deep learning. Because its statistics come from the mini-batch itself, its behavior depends on the batch: BN is a batch-level operation, not a per-sample one.

Recently, Mounayer et al. [1] introduced **Rank Reduction Autoencoders** (RRAEs) and showed, theoretically and empirically, that the truncated SVD can act as a powerful regularizer for autoencoders without optimizing any additional loss. Like BN, the truncated SVD is a batch-level operation that changes the statistics of the batch: it constrains the batch to lie in a low-dimensional subspace spanned by its top singular vectors, which is the best rank-k approximation of the batch in the Frobenius norm by the Eckart-Young theorem [3]. Unlike BN, this constraint is geometric (a rank constraint) rather than moment-based (a mean and variance constraint).

In the RRAE architecture, the SVD acts at the autoencoder bottleneck and its effect is entangled with the rest of the design. The question this project asks is simpler and has not been studied separately: **can the truncated SVD alone normalize a batch as well as BN does?** To answer it, we package the operation into a standalone **SVD layer** (also called the RR layer below) that can be placed anywhere BN can, and we compare Plain, BN, and SVD-layer networks under identical conditions on a supervised localization task where the layer plays BN's usual role inside a feature extractor (Section 4).

Our contributions are:

- A standalone, reusable PyTorch **SVD layer** with a per-batch truncated SVD in training mode and a frozen inference basis in evaluation mode, usable inside any `torch.nn` module.
- A controlled comparison of Plain vs BN vs SVD layer from **identical initial weights**, over up to 32 seeds,

sweeping rank, noise, dataset size, and background type.

- A diagnosis of a practical failure mode: training instability that traces to a hand-written analytic SVD gradient rather than to the layer itself.

2 The SVD Layer

2.1 Definition

Let a mini-batch of activations be x with shape (N, *), where N is the batch size and * is any feature shape (for a convolutional feature map, C×H×W). The layer first flattens the features and arranges the batch as a matrix X of shape M×N, one column per sample (M = total feature dimension). It then computes the SVD and keeps the top k singular components:

$$X = U\,\Sigma\,V^T \;\rightarrow\; \hat{X} = U_k\,\Sigma_k\,V_k{}^T$$

The reconstruction $\hat{X}$ is reshaped back to (N, *) and passed on. Every sample in the batch is therefore rewritten as a combination of the same k basis vectors U_k: the batch is projected onto its own dominant k-dimensional subspace. The rank k plays the role of a normalization strength dial: the directions are found automatically by the SVD, only the strength (how many directions survive) is chosen by the user.

In terms of imposed statistics: BN forces each feature's batch distribution to have zero mean and unit variance; the truncated SVD instead forces the batch's second-moment structure to be rank-k, i.e. the Gram matrix $\hat{X}^T\hat{X} = V_k\Sigma_k{}^2V_k{}^T$ has exactly k nonzero eigenvalues, the squared singular values. The singular values themselves are not constrained, a point we return to in the conclusion.

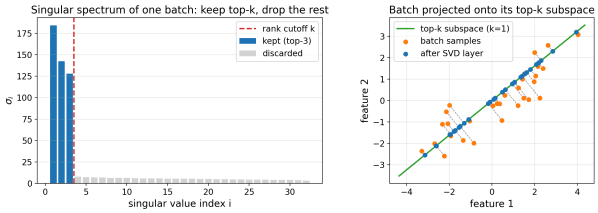


Figure 1: The batch, rebuilt from only its top-k singular components (right: k = 1 example).

2.2 Train and Evaluation Modes

Like BN, the layer needs a batch-independent behavior at test time:

- **Training mode:** a truncated SVD is computed on every mini-batch and the reconstruction is used directly. Gradients flow through the SVD. The basis U_k of each batch is stored in a small rolling bank (the last 20 bases).
- **Evaluation mode:** the stored bases are concatenated and a final SVD extracts a single frozen **inference basis**; test inputs are projected onto it ($\hat{X} = U_kU_k{}^TX$). This mirrors BN's running statistics. The basis can alternatively be computed in one pass over a large data batch (`finalize_basis_from_data`), which is closer to a true full-dataset SVD.

An ablation of the evaluation modes (Section 4.4) shows they matter little on our task: per-batch SVD, basis-bank, and full-data basis give test MSE within a few 10^{-4} of each other, far below every model-to-model difference we report. The simple basis bank is a sound inference procedure.

2.3 Gradients through the SVD

Our first implementation used a hand-written analytic backward pass for the SVD (the standard formula involving factors $1/(\sigma_i-\sigma_j)$). With this backward, networks with an SVD layer after every convolution diverged: training loss climbed to values in the hundreds within a few epochs (Section 4.5). The cause is the $1/(\sigma_i-\sigma_j)$ terms, which blow up whenever two singular values come close, a frequent event for structured feature maps. Replacing the custom backward with PyTorch's native autograd through `torch.linalg.svd` (with the more robust `gesvd` driver on CUDA and a CPU LAPACK fallback) made per-layer training stable. All final results use the native backend.

3 Experimental Setup

3.1 Task and Data

All experiments use the same controlled data family: a single isotropic 2D Gaussian source of width σ = 0.2 on a $[-1, 1]^2$ grid rendered as a 64×64 image, with the center (x_0, y_0) drawn uniformly in $[-0.5, 0.5]^2$. The ground-truth image for any center is analytic and the data has exactly two generative degrees of freedom, so the task is fully controlled. Images are standardized to zero mean and unit variance. Each seed generates its own train/test split from a dedicated RNG, so averaging over seeds averages over both initialization and data draw.

The task is to regress the center from the image: point-source localization as it appears in astrometry and single-molecule microscopy, a smooth, low-rank signal whose position must be recovered, optionally under noise or on top of background clutter.

3.2 Architectures for the Localization Study

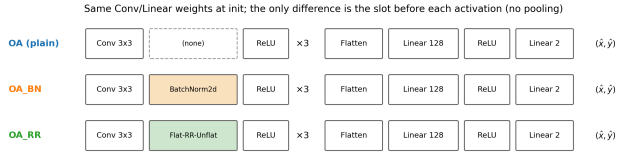


Figure 2: The three architectures. Identical Conv/Linear weights; only the slot between each convolution and its ReLU changes (nothing / BatchNorm2d / SVD layer).

The backbone is 3 convolutional blocks (3×3 kernels, padding 1, channels 1 → 8 → 16 → 32, no pooling, so the spatial size stays 64×64), followed by a plain regression head Linear(32·64·64 → 128) → ReLU → Linear(128 → 2). Three variants differ only in the slot between each convolution and its activation:

- **OA** (plain): conv → ReLU.
- **OA_BN**: conv → BatchNorm2d → ReLU.
- **OA_RR**: conv → SVD layer (flatten, truncated SVD, unflatten) → ReLU.

Before constructing each variant, the same seed is re-applied to the RNG, so all three share **identical initial Conv and Linear weights**; the only difference is the normalization slot. Unless stated otherwise: rank k = 8, batch size 32, Adam [4] with learning rate 10^{-3}, MSE loss, 20 epochs at N_{tr} = 800 with epochs scaled up proportionally as N_{tr} shrinks (so every configuration gets a comparable gradient-update budget), and up to 32 seeds per reported point.

3.3 Experiments

- **Rank sweep** (clean data, per-layer placement, 4 seeds): test error as a function of k ∈ {2, 4, 8, 16, 32}.
- **Small data + heavy noise** (per-layer placement, 32 seeds): additive Gaussian pixel noise with standard deviation 1.0 on train and test images (the signal peak is 1.0, so pixel SNR is at or below 1), N_{tr} swept over {25, 50, 100, 200, 400}. This is the headline experiment.
- **Real backgrounds via source injection** (per-layer placement, 16 seeds): a synthetic source with known position is injected onto real image backgrounds (MNIST digits scaled to 64×64, amplitude 1.5× the source peak; disjoint background pools for train and test). Source injection is astronomy's standard external-validity check: the ground truth stays exact while the clutter statistics become real.

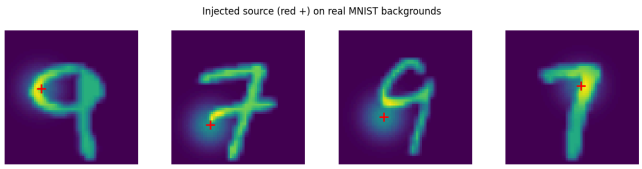


Figure 3: Injected source (red +) on real MNIST backgrounds. The label stays exact; the clutter is real and structured.

4 Results

4.1 Rank Sweep

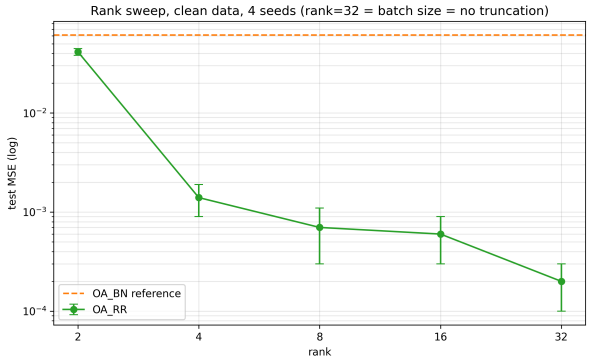


Figure 4: Test MSE (log scale) vs rank k on clean data (per-layer placement, 4 seeds); rank 32 equals the batch size, i.e. no truncation. Dashed line: per-layer BN reference on the same data (0.061); the plain network reaches below 10^{-4}.

Test error falls monotonically as k grows: mean test MSE 0.0414 ± 0.0033 at k = 2, 0.0014 at k = 4, 0.0007 at k = 8, 0.0006 at k = 16, and 0.0002 at k = 32, the batch size, where the layer no longer truncates during training. k = 2 is too aggressive for a supervised feature extractor, costing about 60× relative to k = 8: the batch subspace must represent both coordinates of the center plus nuisance variation, and two directions are not enough. From k = 8 on, the layer is within a factor of about 3 of the no-truncation point. Notably, even the worst rank setting (k = 2, 0.0414) still beats the per-layer BN reference on the same clean data (0.0611). We fix k = 8 for the rest of this section as the smallest rank that solves the task comfortably.

4.2 Small Data + Heavy Noise: The SVD Layer Wins Outright

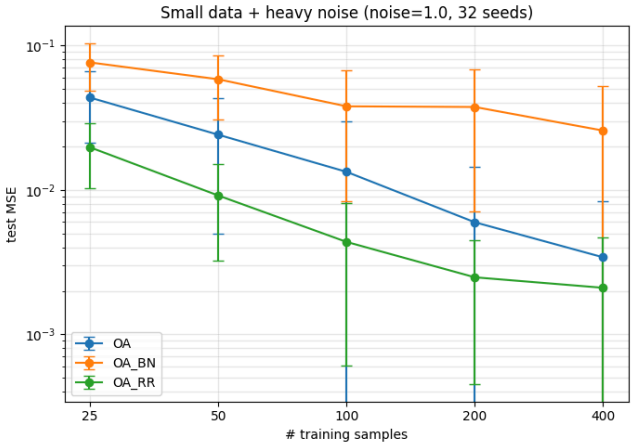


Figure 5: Test MSE (log scale) vs training-set size under heavy input noise (std 1.0), 32 seeds, mean ± std. The SVD layer (OA_RR, green) beats both BN (orange) and the plain network (blue) at every N_{tr}.

N_{tr}	OA (plain)	OA_BN	OA_RR (SVD)	RR vs OA	RR vs BN
25	0.0436 ± 0.0224	0.0763 ± 0.0276	**0.0197 ± 0.0095**	2.2×	3.9×
50	0.0241 ± 0.0192	0.0584 ± 0.0276	**0.0091 ± 0.0059**	2.6×	6.4×
100	0.0133 ± 0.0165	0.0379 ± 0.0296	**0.0044 ± 0.0038**	3.0×	8.6×
200	0.0060 ± 0.0085	0.0375 ± 0.0304	**0.0025 ± 0.0020**	2.4×	15.0×
400	0.0034 ± 0.0049	0.0258 ± 0.0263	**0.0021 ± 0.0026**	1.6×	12.3×

Table 1: Test MSE (mean ± std over 32 seeds) under input noise std 1.0. Last two columns: factor by which the SVD layer improves on the plain network and on BN.

This is the headline result. Under heavy input noise the per-layer SVD network beats **both** baselines at every training-set size from 25 to 400 samples, by 1.6–3.0× over the plain network and 3.9–15× over BN. It also has the tightest seed variance of the three at every point and the smallest generalization gap: its test-minus-train gap falls to the 10^{-3} level from N_{tr} = 100 on, while the plain network drives train error to zero by memorizing the noise, so its gap is essentially its whole test error. The mechanism is exactly the intended one: i.i.d. pixel noise is spread across all singular directions, while the smooth source occupies a low-rank subspace, so the top-k projection strips noise faster than signal at every layer.

BN is the worst of the three in this regime. It trains (train MSE around 0.02–0.06) but generalizes poorly, and its across-seed variance is the largest. The SVD layer is therefore not just "a BN alternative that survives noise": on this task family it is a stronger regularizer than BN wherever we compared them.

4.3 Real Backgrounds: The Failure Mode, Localized

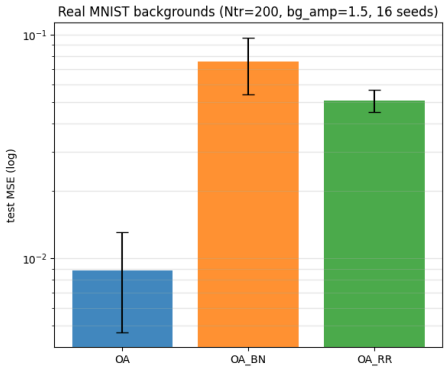


Figure 6: Test MSE (log scale) on real MNIST backgrounds (N_{tr} = 200, background amplitude 1.5, 16 seeds). Plain wins; the SVD layer still beats BN.

	OA (plain)	OA_BN	OA_RR (SVD)
Test MSE	**0.0089 ± 0.0042**	0.0755 ± 0.0215	0.0509 ± 0.0057

Table 2: Source injection on real MNIST backgrounds, mean ± std over 16 seeds.

With real, structured clutter the ranking changes: the plain network is best (0.0089), the SVD layer second (0.0509), BN worst (0.0755). The SVD layer still beats BN by 1.5×, but it loses to the plain network by about 5.7×. The reason

is structural: MNIST digits are themselves approximately low-rank and their amplitude (1.5× the source peak) makes them the dominant part of the batch's variance. Keeping the top singular components therefore keeps the background as faithfully as the signal, so the projection no longer preferentially removes the nuisance. The layer's advantage is specific to nuisance that is spread thinly across many directions (noise-like), and inverts when the nuisance is concentrated in a few strong directions (structure-like).

4.4 Where Does the Layer's Residual Error Come From?

On clean data with a single SVD layer (k = 8, 4 seeds), we evaluated the same trained models three ways: per-batch SVD (training behavior), the default basis-bank inference basis, and a basis computed from one large pass over the data. Mean test MSE was 0.0004, 0.0007, and 0.0006 respectively. All three agree to within a few 10^{-4}, far below every model-to-model difference in Tables 1 and 2, so the frozen-basis approximation costs little on this task; the layer's residual error relative to the plain network comes from the rank constraint itself, not from the train/eval mismatch.

4.5 Training Stability and the SVD Gradient

Our first per-layer runs diverged: with the hand-written analytic SVD backward, training loss climbed by orders of magnitude within a few epochs and never recovered. We initially misread this as "the SVD layer cannot be stacked". It is not: the instability disappears entirely when gradients are taken through PyTorch's native SVD autograd. With the native backend, the same per-layer network trains smoothly on clean data (training loss 2.68 → $3 \cdot 10^{-4}$ over 20 epochs), and over 8 seeds at N_{tr} = 800 reaches test MSE 0.0003 ± 0.0002, between the plain network (0.0000) and BN (0.0009 ± 0.0004), and it is this configuration that produces the results of Table 1. The practical lesson for anyone reusing the layer is that the backward pass through the SVD is the numerically dangerous part, and the battle-tested library implementation should be preferred over textbook analytic formulas.

5 Discussion

When the layer wins and when it loses, one geometry explains both. The SVD layer is a regularizer that acts on directions: it wins exactly when the nuisance is high-rank (i.i.d. noise spread thinly across all singular directions) while the signal is low-rank, and it loses when the nuisance is itself low-rank and strong (real backgrounds), because then the top-k projection preserves the clutter as faithfully as the signal. In both cases the behavior is the Eckart-Young projection doing exactly what it says: keep the dominant directions, discard the rest. The layer is therefore not a universal drop-in improvement but a prior: use it when nuisance is noise-like, avoid it when nuisance is structure-like.

Why BN loses everywhere here. BN equalizes the scale of each coordinate without changing which directions carry information. On this task the useful structure is a question of directions, not of scale: under input noise, rescaling every feature to unit variance amplifies exactly the noise-dominated features the SVD layer discards. This is consistent with the optimization-centric accounts of BN's benefits [5, 6], which concern deep, hard-to-optimize networks; the networks here are shallow and easy to optimize, so BN's costs show while its benefits do not. Our claim is scoped accordingly: BN lost in every comparison we ran on this task family, which is not a statement about BN at ImageNet scale.

Choosing the rank. The rank is the layer's single hyperparameter and acts as a strength dial: the directions are found automatically by the SVD, only how many survive is chosen by the user. The sweep of Section 4.1 shows the cost surface is forgiving on one side and steep on the other: too large a rank degrades gracefully toward the unconstrained network, while too small a rank (k = 2, below what the task's signal plus nuisance requires) costs 60×. When in doubt, err on the large side.

6 Conclusion and Future Work

We isolated the truncated SVD from the RRAE architecture into a standalone, reusable PyTorch layer and compared it head-to-head with Batch Normalization under controlled conditions (identical initial weights, up to 32 seeds). Our findings:

- The SVD layer **beat BN in every comparison we ran** (clean data, noisy data, small training sets, real backgrounds), by 1.5–15×.
- Under heavy input noise it beat **even the unregularized network at every dataset size tested** (1.6–3×), with the tightest seed variance and the smallest train/test gap. In this regime it is a genuinely better drop-in choice than BN.
- Its weakness is equally clear: when the nuisance is itself low-rank and strong (real structured backgrounds), keeping the top singular components preserves the clutter, and the plain network wins.
- Training instability that at first appeared intrinsic to stacking SVD layers was a bug in a hand-written SVD gradient; with native autograd the layer trains stably at every position in the network.

Future work. The layer constrains the rank of the batch but not its scale: the retained singular values in Σ can be arbitrarily large or small, so, unlike BN, the mean and variance of the batch are not pushed toward fixed values. Normalizing the retained singular values (encouraging them toward 1, or shrinking the discarded tail toward 0 softly rather than by hard truncation) would make the layer a true

normalizer in BN's sense and is the natural next step. A second direction is the structured-background failure: reversing the selection (rejecting the top components, in the spirit of robust-PCA and KLIP-style background subtraction) or choosing the rank automatically from an energy threshold could let the same machinery handle structure-like nuisance as well as noise-like nuisance. Finally, all conclusions concern one controlled synthetic task family; the natural escalation is real data at larger scale, where the noise-like vs structure-like character of the nuisance is not known in advance.

7 Code Availability

The source package (mirrored at `github.com/Yousef10p/RR-Layer`) contains `svd_layer.py`, a self-contained implementation of the layer (`RRLayer`) with the train/eval behavior of Section 2.2, input validation, both SVD backends (native autograd by default), and a runnable self-check. It can be dropped into any module exactly like BN:

```
nn.Sequential(nn.Conv2d(8, 16, 3, padding=1),
              RRLayer(rank=8), nn.ReLU())
```

The package also contains the canonical experiment notebook that produced Tables 1 and 2 and all result figures, plus a README with instructions to run everything top to bottom on a GPU runtime. The reference implementation of the layer is our mentor's `RR_layer` repository [7], now also available as a packaged library (`pip install RR_layer`).

References

[1] J. Mounayer, S. Rodriguez, C. Ghnatios, C. Farhat, and F. Chinesta. Rank Reduction Autoencoders. arXiv preprint arXiv:2405.13980, 2024.

[2] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, pp. 448–456, 2015.

[3] C. Eckart and G. Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, 1936.

[4] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

[5] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry. How Does Batch Normalization Help Optimization? In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[6] N. Bjorck, C. P. Gomes, B. Selman, and K. Q. Weinberger. Understanding Batch Normalization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[7] J. Mounayer. RR_layer: reference implementation of the Rank Reduction layer. `github.com/JadM133/RR_layer`, accessed 2026.