Memorization in Overparameterized Autoencoders

Adityanarayanan Radhakrishnan [1] **Mikhail Belkin** [2] **Caroline Uhler** [1]

Abstract

Interpolation of data in deep neural networks has become a subject of significant research interest. We prove that over-parameterized single layer fully connected autoencoders do not merely interpolate, but rather, memorize training data: they produce outputs in (a non-linear version of) the span of the training examples. In contrast to fully connected autoencoders, we prove that depth is necessary for memorization in convolutional autoencoders. Moreover, we observe that adding nonlinearity to deep convolutional autoencoders results in a stronger form of memorization: instead of outputting points in the span of the training images, deep convolutional autoencoders tend to output *individual training images*. Since convolutional autoencoder components are building blocks of deep convolutional networks, we envision that our findings will shed light on the important question of the inductive bias in over-parameterized deep networks.

1. Introduction

As deep convolutional neural networks (CNNs) become ubiquitous in computer vision thanks to their strong performance on a range of tasks (Goodfellow et al., 2016), recent work has begun to analyze the role of *interpolation* (perfectly fitting training data) in such networks (Arpit et al., 2017; Zhang et al., 2017). These works show that deep overparametrized networks can interpolate training data even when the labels are random. For an overparameterized model, there are typically infinitely many interpolating solutions. Thus it is important to characterize the inductive bias of an algorithm, i.e., the properties of the specific solution chosen by the training procedure.

In this paper we study *autoencoders* (Goodfellow et al., 2016), i.e. maps $\psi : \mathbb{R}^d \to \mathbb{R}^d$ that are trained to satisfy

$$\psi(x^{(i)}) \approx x^{(i)} \text{ with } x^{(i)} \in \mathbb{R}^d, 1 \leq i \leq n.$$

Autoencoders are typically trained by solving

$$\arg\min_{\psi \in \Psi} \sum_{i=1}^{n} \|\psi(x^{(i)}) - x^{(i)}\|^2$$

by gradient descent over a parametrized function space Ψ. There are many interpolating solutions to the autoencoding problem in the overparametrized setting. We characterize the inductive bias as *memorization* when the autoencoder output is within the span of the training data and *strong memorization* when the output is close to one of the training examples for almost any input.

Studying memorization in the context of autoencoders is relevant since (1) components of convolutional autoencoders are building blocks of many CNNs; (2) layerwise pre-training using autoencoders is a standard technique to initialize individual layers of CNNs to improve training (Belilovsky et al., 2019; Bengio et al., 2007; Erhan et al., 2010); and (3) autoencoder architectures are used in many image-to-image tasks such as image segmentation, image impainting, etc. (Ulyanov et al., 2017). While the results in this paper hold generally for autoencoders, we concentrate on image data, since this allows identifying memorization by visual inspection of the input and output.

To illustrate the memorization phenomenon, consider linear single layer fully connected autoencoders. This autoencoding problem can be reduced to linear regression (see Appendix A). It is well-known that solving overparametrized linear regression by gradient descent initialized at zero converges to the minimum norm solution (see, e.g., Theorem 6.1 in (Engl et al., 1996)). This minimum norm solution translated to the autoencoding setting corresponds to memorization of the training data: after training the autoencoder, any input image is mapped to an image that lies in the span of the training set.

In this paper, we prove that the memorization property extends to nonlinear single layer fully connected autoencoders.

We proceed to show that memorization extends to deep

[1] Laboratory for Information & Decision Systems, and Institute for Data, Systems, and Society, Massachusetts Institute of Technology [2] Department of Computer Science and Engineering, Ohio State University. Correspondence to: Adit Radhakrishnan <aradha@mit.edu>, Mikhail Belkin <mbelkin@cse.ohio-state.edu>, Caroline Uhler <cuhler@mit.edu>.

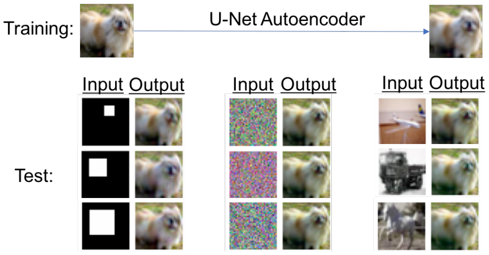


Figure 1. U-Net Autoencoder trained on a single image from CIFAR10 for 2000 iterations. When fed random sized white squares, a standard Gaussian, or new images from CIFAR10 the model outputs the training image.

(but not shallow) convolutional autoencoders. As a striking illustration of this phenomenon consider Figure 1. After training a U-Net architecture (Ronneberger et al., 2015), which is commonly used in image-to-image tasks (Ulyanov et al., 2017), on a single training image, any input image is mapped to the training image. Related ideas were concurrently explored for autoencoders trained on a single example in (Zhang et al., 2019).

The main contributions of this paper are as follows. Building on the connection to linear regression, we prove that single layer fully connected nonlinear autoencoders produce outputs in the "nonlinear" span (see Definition 2) of the training data. Interestingly, we show in Section 3 that in contrast to fully connected autoencoders, shallow *convolutional* autoencoders do not memorize training data, even when adding filters to increase the number of parameters. In Section 4, we observe that our memorization results for linear CNNs carry over to nonlinear CNNs. Further, nonlinear CNNs demonstrate a strong form of memorization: the trained network outputs individual training images rather than just combinations of training images. We end with a short discussion in Section 5. Appendices E, F, G, and H provide additional details concerning the effect of downsampling, early stopping, and initialization on memorization in linear and nonlinear convolutional autoencoders.

2. Memorization in Single Layer Nonlinear Fully Connected Autoencoders

In this section, we characterize memorization properties of nonlinear single layer fully connected autoencoders initialized at zero. A nonlinear single layer fully connected autoencoder satisfies $\phi(Ax^{(i)}) \approx x^{(i)}$ for $1 \leq i \leq n$, where ϕ is a non-linear function (such as the sigmoid function) that acts element-wise with $x_j^{(i)} \in \text{range}(\phi)$, where $x_j^{(i)}$ denotes the j^{th} element of $x^{(i)}$ for $1 \leq i \leq n$, $1 \leq j \leq d$.

In the following, we provide a closed form solution for the matrix A when initialized at $A^{(0)} = \mathbf{0}$ and computed using gradient descent on the mean squared error loss, i.e.

$$\min_{A \in \mathbb{R}^{d \times d}} \frac{1}{2} \sum_{i=1}^{n} (x^{(i)} - \phi(Ax^{(i)}))^T (x^{(i)} - \phi(Ax^{(i)})). \quad (1)$$

Let $\phi^{-1}(y)$ be the pre-image of $y \in \mathbb{R}$ of minimum ℓ_2 norm and for each $j \in \{1, 2, \dots d\}$ let

$$x'_j = \arg\max_{1 \leq i \leq n} |\phi^{-1}(x_j^{(i)})|.$$

In the following, we provide three mild assumptions that are often satisfied in practice under which a closed form formula for A can be derived in the nonlinear overparameterized setting.

Assumption 1. *For all $j \in \{1, 2, \dots, d\}$ it holds that*

(a) $0 < x_j^{(i)} < 1$ for all $1 \leq i \leq n$;

(b) $x_j^{(i)} < \phi(0)$ (or $x_j^{(i)} > \phi(0)$) for all $1 \leq i \leq n$;

(c) ϕ satisfies one of the following conditions:

(1) if $\phi^{-1}(x'_j) > 0$, then ϕ is strictly convex and monotonically decreasing on $[0, \phi^{-1}(x'_j)]$;

(2) if $\phi^{-1}(x'_j) > 0$, then ϕ is strictly concave and monotonically increasing on $[0, \phi^{-1}(x'_j)]$;

(3) if $\phi^{-1}(x'_j) < 0$, then ϕ is strictly convex and monotonically increasing on $[\phi^{-1}(x'_j), 0]$;

(4) if $\phi^{-1}(x'_j) < 0$, then ϕ is strictly concave and monotonically decreasing on $[\phi^{-1}(x'_j), 0]$.

Assumption (a) typically holds for un-normalized images. Assumption (b) is satisfied for example when using a min-max scaling of the images. Assumption (c) holds for many nonlinearities used in practice including the sigmoid and tanh functions.

To prove memorization for overparametrized nonlinear single layer fully connected autoencoders, we first show how to reduce the non-linear setting to the linear setting.

Theorem 1. *Let $n < d$ (overparametrized setting). Under Assumption 1, solving (1) to achieve $\phi(Ax^{(i)}) \approx x^{(i)}$ using a variant of gradient descent (with an adaptive learning rate as described in Supplementary Material B) initialized at $A^{(0)} = \mathbf{0}$ converges to a solution $A^{(\infty)}$ that satisfies the linear system $A^{(\infty)}x^{(i)} = \phi^{-1}(x^{(i)})$ for all $1 \leq i \leq n$.*

The proof is presented in Supplementary Material B. Given our empirical observations using a constant learning rate, we suspect that the adaptive learning rate used for gradient descent in the proof is not necessary for the result to hold.

As a consequence of Theorem 1, the single layer nonlinear autoencoding problem can be reduced to a linear regression problem. This allows us to define a memorization property for nonlinear systems by introducing nonlinear analogs of an eigenvector and the span.

Definition 1 (ϕ-eigenvector)**.** *Given a matrix $A \in \mathbb{R}^{d\times d}$ and element-wise nonlinearity ϕ, a vector $u \in \mathbb{R}^d$ is a ϕ-eigenvector of A with ϕ-eigenvalue λ if $\phi(Au) = \lambda u$.*

Definition 2 (ϕ-span)**.** *Given a set of vectors $U = \{u_1, \dots u_r\}$ with $u_i \in \mathbb{R}^d$ and an element-wise nonlinearity ϕ, let $\phi^{-1}(U) = \{\phi^{-1}(u_1)\dots\phi^{-1}(u_r)\}$. The nonlinear span of U corresponding to ϕ (denoted $\phi\text{-span}(U)$) consists of all vectors $\phi(v)$ such that $v \in \text{span}(\phi^{-1}(U))$.*

The following corollary characterizes memorization for nonlinear single layer fully connected autoencoders.

Corollary (Memorization in non-linear single layer fully connected autoencoders)**.** *Let $n < d$ (overparametrized setting) and let $A^{(\infty)}$ be the solution to (1) using a variant of gradient descent with an adaptive learning rate initialized at $A^{(0)} = \mathbf{0}$. Then under Assumption 1, $\text{rank}(A^{(\infty)}) = \dim(\text{span}(X))$; in addition, the training examples $x^{(i)}$, $1 \le i \le n$, are ϕ-eigenvectors of $A^{(\infty)}$ with eigenvalue 1 and $\phi(A^{(\infty)}y) \in \phi\text{-span}(X)$ for any $y \in \mathbb{R}^d$.*

Proof. Let S denote the covariance matrix of the training examples and let $r := \text{rank}(S)$. It then follows from Theorem 1 and the minimum norm solution of linear regression that $\text{rank}(A^{(\infty)}) \le r$. Since in the overparameterized setting, $A^{(\infty)}$ achieves 0 training error, the training examples satisfy $\phi(A^{(\infty)}x^{(i)}) = x^{(i)}$ for all $1 \le i \le n$, which implies that the examples are ϕ-eigenvectors with eigenvalue 1. Hence, it follows that $\text{rank}(A^{(\infty)}) \ge r$ and thus $\text{rank}(A^{(\infty)}) = r$. Lastly, since the ϕ-eigenvectors are the training examples, it follows that $\phi(A^{(\infty)}y) \in \phi\text{-span}(X)$ for any $y \in \mathbb{R}^d$. □

3. Role of Depth for Memorization in Convolutional Autoencoders

In contrast to single layer autoencoders discussed in the previous section, we now show that shallow linear convolutional autoencoders in general do not memorize training data even in the overparametrized setting; hence depth is necessary for memorization in convolutional autoencoders.

For the following discussion of convolutional autoencoders, let the training samples be images in $\mathbb{R}^{s\times s}$. While all our results also hold for color images, we dropped the color channel to simplify notation.

Theorem 2. *A single filter convolutional autoencoder with kernel size k and $\frac{k-1}{2}$ zero padding trained to autoencode an image $x \in \mathbb{R}^{s\times s}$ using gradient descent on the mean squared error loss learns a rank s^2 solution.*

The proof is presented in Supplementary Material C. The main ingredient of the proof is the construction of the matrix A to represent a linear convolutional autoencoder. An algorithm for obtaining A for any linear convolutional autoencoder is presented in Supplementary Material D. Theorem 2 implies that even in the overparameterized setting, a single layer single filter convolutional autoencoder will not memorize training data. For example, a network with a kernel of size 5 and a single training image of size $s = 2$ is overparametrized, since the number of parameters is 25 while the input has dimension 4. However, in contrast to the non-convolutional setting, Theorem 2 implies that the rank of the learned solution is 4, which exceeds the number of training examples; i.e., memorization does not occur.

As explained in the following, this contrasting behavior stems from the added constraints imposed on the matrix A through convolutions, in particular the zeros forced by the structure of the matrix. A concrete example illustrating this constraint is provided in Supplementary Material D. We now prove that these forced zeros prevent memorization in single layer single filter convolutional autoencoders. The following lemma shows that a single layer matrix with just one forced zero cannot memorize arbitrary inputs.

Lemma 1. *A single layer linear autoencoder, represented by a matrix $A \in \mathbb{R}^{d\times d}$ with a single forced zero entry cannot memorize an arbitrary $v \in \mathbb{R}^d$.*

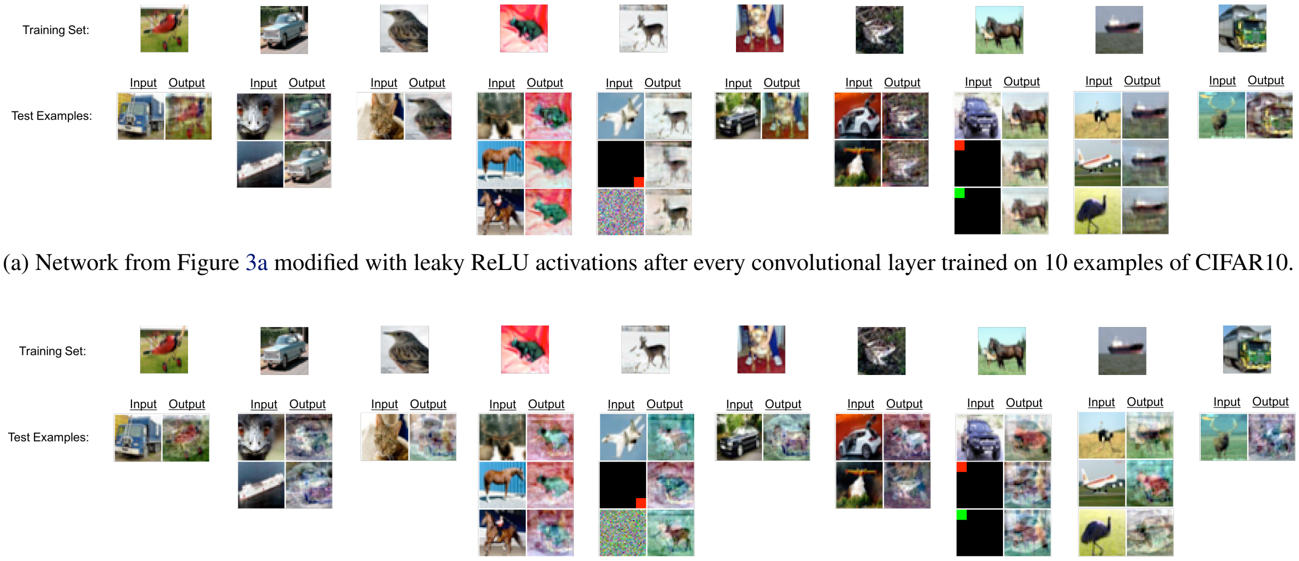
The proof follows directly from the fact that in the linear setting, memorization corresponds to projection onto the training example and thus cannot have a zero in a fixed data-independent entry. Since single layer single filter convolutional autoencoders have forced zeros, Lemma 1 shows that these networks cannot memorize arbitrary inputs. Next, we show that shallow convolutional autoencoders still contain forced zeros regardless of the number of filters that are used in the intermediate layers.

Theorem 3. *At least $s - 1$ layers are required for memorization (regardless of the number of filters per layer) in a linear convolutional autoencoder with filters of kernel size 3 applied to images of size $s \times s$.*

This lower bound follows by analyzing the forced zero pattern of A^{s-1}, which corresponds to the operator for the $s - 1$ layer network. Importantly, Theorem 3 shows that adding filters cannot make up for missing depth, i.e., overparameterization through depth rather than filters is necessary for memorization in convolutional autoencoders. The following corollary emphasizes this point.

Corollary. *A 2-layer linear convolutional autoencoder with filters of kernel size 3 and stride 1 for the hidden representation cannot memorize images of size 4×4 or larger, independently of the number of filters.*

This shows that depth is necessary for memorization in convolutional autoencoders. In Appendix E, we provide empirical evidence that depth is sufficient for memorization, and refine the lower bound from 3 to a lower bound



(a) Network from Figure 3a modified with leaky ReLU activations after every convolutional layer trained on 10 examples of CIFAR10.

(b) Network from Figure 3a trained on 10 examples of CIFAR10.

Figure 2. A comparison of memorization between (a) nonlinear and (b) linear convolutional autoencoders. When trained on 10 examples, one from each class of CIFAR10, the nonlinear autoencoder demonstrates a stronger form of memorization by outputting specific training examples instead of linear combinations of training examples when applied to new inputs.

of $\lceil \frac{s^4}{9} \rceil$ layers needed to identify memorization in linear convolutional autoencoders. While the number of layers needed for memorization are large according to this lower bound, in Appendix F, we show empirically that *downsampling* through strided convolution allows a network to memorize with far fewer layers.

4. Strong Memorization in Nonlinear Convolutional Autoencoders

We now provide evidence that our observations regarding memorization in linear convolutional autoencoders extend to the nonlinear setting. In Figure 2a, we observe that a downsampling nonlinear convolutional autoencoder with leaky ReLU activations (described in Figure 3a) *strongly memorizes* 10 examples, one from each class of CIFAR10. That is, given a new test example from CIFAR10, samples from a standard Gaussian, or random sized color squares, the model outputs an image visually similar to one of the training examples instead of a combination of training examples. This is in contrast to deep linear convolutional autoencoders; for example, in Figure 2b, we see that training the linear model from 3a leads to the model outputting linear combinations of the training examples. These results suggest that for deep nonlinear convolutional autoencoders the training examples are strongly attractive fixed points.

5. Conclusions and Future Work

This paper identified the mechanism behind memorization in autoencoders. While it is well-known that linear regression converges to a minimum norm solution when initialized at zero, we tied this phenomenon to memorization in non-linear single layer fully connected autoencoders, showing that they produce output in the nonlinear span of the training examples. Furthermore, we showed that convolutional autoencoders behave quite differently since not every overparameterized convolutional autoencoder memorizes. Indeed, we showed that overparameterization by adding depth or downsampling is necessary and empirically sufficient for memorization in the convolutional setting, while overparameterization by extending the number of filters in a layer does not lead to memorization.

Interestingly, we observed empirically that the phenomenon of memorization is pronounced in the non-linear setting, where nearly arbitrary input images are mapped to output images that are visually identifiable as one of the training images rather than a linear combination thereof as in the linear setting. While the exact mechanism for this strong form of memorization in the non-linear setting still needs to be understood, this phenomenon is reminiscent of FastICA in Independent Component Analysis (Hyvrinen & Oja, 1997) or more general non-linear eigenproblems (Belkin et al., 2018b), where every "eigenvector" (corresponding to training examples in our setting) of certain iterative maps has its own basin of attraction. We conjecture that increasing the depth may play the role of increasing the number of iterations in those methods.

Since the use of deep networks with near zero initialization is the current standard for image classification tasks, we expect that our memorization results also carry over to these application domains. We note that memorization is a particular form of interpolation (zero training loss) and interpolation has been demonstrated to be capable of generalizing to test data in neural networks and a range of other methods (Zhang et al., 2017; Belkin et al., 2018a). Our work could provide a mechanism to link overparameterization and memorization with generalization properties observed in deep convolutional networks.

Acknowledgements

Adityanarayanan Radhakrishnan was supported by National Science Foundation (DMS-1651995). Caroline Uhler was partially supported by National Science Foundation (DMS-1651995), Office of Naval Research (N00014-17-1-2147 and N00014-18-1-2765), IBM, and a Sloan Fellowship. Mikhail Belkin acknowledges support from NSF (IIS-1815697 and IIS-1631460).

References

Arpit, D., Jastrzebski, S., Ballas, N., Krueger, D., Bengio, E., Kanwal, M. S., Maharaj, T., Fischer, A., Courville, A., Bengio, Y., and Lacoste-Julien, S. A closer look at memorization in deep networks. In *International Conference on Machine Learning (ICML)*, 2017.

Belilovsky, E., Eickenberg, M., and Oyallon, E. Greedy layerwise learning can scale to ImageNet, 2019. arXiv:1812.11446.

Belkin, M., Hus, D., Ma, S., and Mandal, S. Reconciling modern machine learning and the bias-variance tradeoff, 2018a. arXiv:1812.11118.

Belkin, M., Rademacher, L., and Voss, J. Eigenvectors of orthogonally decomposable functions. *SIAM Journal on Computing*, 47(2):547–615, 2018b.

Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. Greedy layer-wise training of deep networks. In *Neural Information Processing Systems (NIPS)*, 2007.

Engl, H. W., Hanke, M., and Neubauer, A. *Regularization of inverse problems*, volume 375. Springer Science & Business Media, 1996.

Erhan, D., Bengio, Y., Courville, A., ne Manzagol, P.-A., Vincent, P., and Bengio, S. Why does unsupervised pretraining help deep learning? *Journal of Machine Learning Research (JMLR)*, 11:625–660, 2010.

Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2010.

Goodfellow, I., Bengio, Y., and Courville, A. *Deep Learning*, volume 1. MIT Press, 2016.

He, K., Zhang, X., Ren, S., and Sun, J. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *International Conference on Computer Vision (ICCV)*, 2015.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Computer Vision and Pattern Recognition (CVPR)*, 2016.

Hyvrinen, A. and Oja, E. A fast fixed-point algorithm for independent component analysis. *Neural Computation*, 9(7):1483–1492, 1997.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

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.

Ronneberger, O., Fischer, P., and Brox, T. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI)*, 2015.

Ulyanov, D., Vedaldi, A., and Lempitsky, V. Deep image prior, 2017. arXiv:1711.10925.

Xu, B., Wang, N., Chen, T., and Li, M. Empirical evaluation of rectified activations in convolution network, 2015. arXiv:1505.00853.

Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. Understanding deep learning requires rethinking generalization. In *International Conference on Learning Representations (ICLR)*, 2017.

Zhang, C., Bengio, S., Hardt, M., and Singer, Y. Identity crisis: Memorization and generalization under extreme overparameterization, 2019. arXiv:1902.04698.

A. Minimum Norm Solution for Linear Fully Connected Autoencoders

In the following, we analyze the solution when using gradient descent to solve the autoencoding problem for the system $Ax^{(i)} = x^{(i)}$ for $1 \leq i \leq n$ with $x^{(i)} \in \mathbb{R}^d$. The loss function is

$$L = \frac{1}{2}\sum_{i=1}^{n}(Ax^{(i)} - x^{(i)})^T(Ax^{(i)} - x^{(i)})$$

and the gradient with respect to the parameters A is

$$\frac{\partial L}{\partial A} = (A - I)\sum_{i=1}^{n}(x^{(i)}x^{(i)^T}).$$

Let $S = \sum_{i=1}^{n}(x^{(i)}x^{(i)^T})$. Hence gradient descent with learning rate $\gamma > 0$ will proceed according to the equation:

$$\begin{aligned} A^{(t+1)} &= A^{(t)} + \gamma(I - A^{(t)})S \\ &= A^{(t)}(I - \gamma S) + \gamma S \end{aligned}$$

Now suppose that $A^{(0)} = \mathbf{0}$, then we can directly solve the recurrence relation for $t > 0$, namely

$$A^{(t)} = I - (I - \gamma S)^t$$

Note that S is a real symmetric matrix, and so it has eigendecomposition $S = Q\Lambda Q^T$ where Λ is a diagonal matrix with eigenvalue entries $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_r$ (where r is the rank of S). Then:

$$\begin{aligned} A^{(t)} &= I - Q(I - \gamma\Lambda)^t Q^T \\ &= Q(I - (I - \gamma\Lambda)^t)Q^T. \end{aligned}$$

Now if $\gamma < \frac{1}{\lambda_1}$, then we have that:

$$A^{(\infty)} = Q\begin{bmatrix} I_{r\times r} & \mathbf{0}_{r\times d-r} \\ \mathbf{0}_{d-r\times r} & \mathbf{0}_{d-r\times d-r}\end{bmatrix}Q^T,$$

which is the minimum norm solution.

B. Proof for Nonlinear Fully Connected Autoencoder

In the following, we present the proof of Theorem 2 from the main text.

Proof. As we are using a fully connected network, the rows of the matrix A can be optimized independently during gradient descent. Thus without loss of generality, we only consider the convergence of the first row of the matrix A denoted $A_1 = [a_1, a_2, \ldots a_d]$ to find $A_1^{(\infty)}$.The loss function for optimizing row A_1 is given by:

$$L = \frac{1}{2}\sum_{i=1}^{n}(x_1^{(i)} - \phi(A_1 x^{(i)}))^2.$$

Our proof involves using gradient descent on L but with a different adaptive learning rate per example. That is, let $\gamma_i^{(t)}$ be the learning rate for training example i at iteration t of gradient descent. Without loss of generality, fix $j \in \{1, \ldots, d\}$. The gradient descent equation for parameter a_j is:

$$a_j^{(t+1)} = a_j^{(t)} + \sum_{i=1}^{n}\gamma_i^{(t)}x_j^{(i)}\phi'(A_1^{(t)}x^{(i)})(x_j^{(i)} - \phi(A_1^{(t)}x^{(i)}))$$

To simplify the above equation, we make the following substitution

$$\gamma_i^{(t)} = -\frac{\gamma_i}{\phi'(A_1^{(t)}x^{(i)})},$$

i.e., the adaptive component of the learning rate is the reciprocal of $\phi'(A_1^{(t)}x^{(i)})$ (which is nonzero due to monotonicity conditions on ϕ). Note that we have included the negative sign so that if ϕ is monotonically decreasing on the region of gradient descent, then our learning rate will be positive. Hence the gradient descent equation simplifies to

$$a_j^{(t+1)} = a_j^{(t)} + \sum_{i=1}^{n}\gamma_i x_j^{(i)}(\phi(A_1^{(t)}x^{(i)}) - x_j^{(i)}).$$

Before continuing, we briefly outline the strategy for the remainder of the proof. First, we will use assumption (c) and induction to upper bound the sequence $(\phi(A_1^{(t)}x^{(i)}) - x_j^{(i)})$ with a sequence along a line segment. The iterative form of gradient descent along the line segment will have a simple closed form and so we will obtain a coordinate-wise upper bound on our sequence of interest $A_1^{(t)}$. Next, we show that our upper bound given by iterations along the selected line segment is in fact a coordinate-wise least upper bound. Then we show that $A_1^{(t)}$ is a coordinate-wise monotonically increasing function, meaning that it must converge to the least upper bound established prior.

Without loss of generality assume, $\phi^{-1}(x_k') > 0$ for $1 \leq k \leq d$. By assumption (c), we have that for $x \in [0, \phi^{-1}(x_j^{(i)})]$,

$$\phi(x) < -\frac{\phi(0) - x_j^{(i)}}{\phi^{-1}(x_j^{(i)})}x + \phi(0),$$

since the right hand side is just the line segment joining points $(0, \phi(0))$ and $(\phi^{-1}(x_j^{(i)}), x_j^{(i)})$, which must be above the function $\phi(x)$ if the function is strictly convex. To simplify notation, we write

$$s_i = -\frac{\phi(0) - x_j^{(i)}}{\phi^{-1}(x_j^{(i)})}.$$

Now that we have established a linear upper bound on ϕ, consider a sequence $B_1^{(t)} = [b_1^{(t)}, \ldots b_d^{(t)}]$ analogous to $A_1^{(t)}$

but with updates:

$$b_j^{(t+1)} = b_j^{(t)} + \sum_{i=1}^{n} \gamma_i x_j^{(i)}(-s_i B_1^{(t)} x^{(i)} + \phi(0) - x_j^{(i)})$$

Now if we let $\gamma_i = \frac{\gamma}{s_i}$, then we have

$$b_j^{(t+1)} = b_j^{(t)} - \sum_{i=1}^{n} \gamma x_j^{(i)}(B_1^{(t)} x^{(i)} - \phi^{-1}(x_j^{(i)})),$$

which is the gradient descent update equation with learning rate γ for the first row of the parameters B in solving $Bx^{(i)} = \phi^{-1}(x^{(i)})$ for $1 \leq i \leq n$. Since gradient descent for a linear regression initialized at 0 converges to the minimum norm solution (see Appendix A), we obtain that $B_1^{(t)} x^{(i)} \in [0, \phi^{-1}(x_1')]$ for all $t \geq 0$ when $B_1^{(0)} = \mathbf{0}$.

Next, we wish to show that $B_j^{(t)}$ is a coordinate-wise upper bound for $A_1^{(t)}$. To do this, we first select L such that $\frac{x_j^{(i)}}{x_k^{(i)}} \leq L$ for $1 \leq i \leq n$ and $1 \leq j, k \leq d$ (i.e. $L \geq 1$).

Then, we proceed by induction to show the following:

1. $b_j^{(t)} > a_j^{(t)}$ for all $t \geq 2$ and $1 \leq j \leq d$.
2. For $C_1^{(t)} = [c_1^{(t)}, \dots c_d^{(t)}] = B_1^{(t)} - A_1^{(t)}$, $\frac{c_l^{(t)}}{c_j^{(t)}} \leq L$ for $1 \leq l, j \leq d$ and for all $t \geq 2$.

To simplify notation, we follow induction for $a_1^{(t)}$ and $b_1^{(t)}$ and by symmetry our reasoning follows for $a_j^{(t)}$ and $b_j^{(t)}$ for $2 \leq j \leq d$.

Base Cases :

1. Trivially we have $a_1^{(1)} = b_1^{(1)} = 0$ and so $A_1^{(0)} x = B_1^{(0)} x = 0$.
2. We have that: $a_1^{(1)} = b_1^{(1)} = \sum_{i=1}^{n} \gamma_i x_1^{(i)}(\phi(0) - x_1^{(i)})$. Hence we have $A_1^{(1)} x = B_1^{(1)} x$.
3. Now for $t = 2$,

$$a_1^{(2)} = a_1^{(1)} + \sum_{i=1}^{n} \gamma_i x_1^{(i)}(\phi(A_1^{(1)} x^{(i)}) - x_1^{(i)})$$

$$b_1^{(2)} = b_1^{(t)} + \sum_{i=1}^{n} \gamma_i x_1^{(i)}(-s_i B_1^{(1)} x^{(i)} + \phi(0) - x_1^{(i)})$$

However, we know that $B_1^{(1)} x^{(i)} \in [0, \phi^{-1}(x_1')]$ and since $A_1^{(1)} = B_1^{(1)}$, $A_1^{(1)} x^{(i)} \in [0, \phi^{-1}(x_1')]$. Hence, $b_1^{(2)} > a_1^{(2)}$ since the on the interval $[0, \phi^{-1}(x_1')]$, ϕ is bounded above by the line segments with endpoints $(0, \phi(0))$ and $(\phi^{-1}(x_1^{(i)}), x_1^{(i)})$. Now for the second component of induction, we have:

$$c_j^{(2)} = \sum_{i=1}^{n} \gamma_i x_j^{(i)}(-s_i B_1^{(1)} x^{(i)} + \phi(0) - \phi(A_1^{(1)} x^{(i)}))$$

$$c_l^{(2)} = \sum_{i=1}^{n} \gamma_i x_l^{(i)}(-s_i B_1^{(1)} x^{(i)} + \phi(0) - \phi(A_1^{(1)} x^{(i)}))$$

To simplify the notation, let:

$$G_i^{(1)} = -s_i B_1^{(1)} x^{(i)} + \phi(0) - \phi(A_1^{(1)} x^{(i)})$$

Thus, we have

$$\frac{c_l^{(2)}}{c_j^{(2)}} = \frac{\sum_{i=1}^{n} \gamma_i x_l^{(i)} G_i^{(1)}}{\sum_{i=1}^{n} \gamma_i x_j^{(i)} G_i^{(1)}} = \frac{\sum_{i=1}^{n} \gamma_i \frac{x_l^{(i)}}{x_j^{(i)} \prod_{p \neq i} x_j^{(p)}} G_i^{(1)}}{\sum_{i=1}^{n} \gamma_i \frac{1}{\prod_{p \neq i} x_j^{(p)}} G_i^{(1)}}$$

$$\leq \frac{\sum_{i=1}^{n} \gamma_i L \frac{1}{\prod_{p \neq i} x_j^{(p)}} G_i^{(1)}}{\sum_{i=1}^{n} \gamma_i \frac{1}{\prod_{p \neq i} x_j^{(p)}} G_i^{(1)}} = L$$

Inductive Hypothesis: We now assume that for $t = k$, $b_1^{(k)} > a_1^{(k)}$ and so $B_1^{(k)} x > A_1^{(k)} x$. We also assume $\frac{c_i^{(k)}}{c_j^{(k)}} \leq L$.

Inductive Step: Now we consider $t = k + 1$. Since $b_1^{(k)} = a_1^{(k)} + c_1^{(k)}$ for $c_1^{(k)} > 0$ and since $x_j^{(i)} \in (0, 1)$ for all i, j, we have $B_1^{(k)} x^{(i)} = A_1^{(k)} x^{(i)} + \sum_{j=1}^{d} c_j^{(k)} x_j^{(i)}$.

Consider now the difference between $b_1^{(k+1)}$ and $a_1^{(k+1)}$:

$$\begin{aligned}
c_1^{(k+1)} &= c_1^{(k)} + \sum_{i=0}^{n} \gamma_i x_1^{(i)} (-s_i B_1^k x^{(i)} + \phi(0) - \phi(A_1^{(k)} x^{(i)})) \\
&= c_i^{(k)} + \sum_{i=0}^{n} \gamma_i x_1^{(i)} (-s_i A_1^{(k)} x^{(i)} + \phi(0) - \phi(A_1^{(k)} x^{(i)})) \\
&\qquad - \sum_{i=0}^{n} \gamma_i x_1^{(i)} s_i \sum_{j=1}^{d} c_j^{(k)} x_j^{(i)} \\
&> c_1^{(k)} - \sum_{i=0}^{n} \gamma_i x_1^{(i)} s_i \sum_{j=1}^{d} c_j^{(k)} x_j^{(i)} \\
&\geq c_1^{(k)} - \sum_{i=0}^{n} \gamma_i s_i \sum_{j=1}^{d} c_j^{(k)} \\
&= c_1^{(k)} - \sum_{i=0}^{n} \gamma_i s_i c_1^{(k)} \sum_{j=1}^{d} \frac{c_j^{(k)}}{c_1^{(k)}} \\
&\geq c_i^{(k)} - \sum_{i=0}^{n} \gamma_i s_i c_1^{(k)} L d,
\end{aligned}$$

where the first inequality comes from the fact that $-sA_1^{(k)} x^{(i)} + \phi(0)$ is a point on the line that upper bounds ϕ on the interval $[0, \phi^{-1}(x_1')]$, and the second inequality comes from the fact that each $x_j^{(i)} < 1$. Hence, with a learning rate of

$$\gamma_i = \frac{\gamma}{s_i} \text{ with } \gamma < \frac{1}{nLd},$$

we obtain that $c_1^{(k+1)} = b_1^{(k+1)} - a_1^{(k+1)} > 0$ as desired. Hence, the first component of the induction is complete. To fully complete the induction we must show that $\frac{c_i^{(k+1)}}{c_j^{(k+1)}} \leq L$ for $1 \leq l, j \leq d$. We proceed as we did in the base case:

$$\begin{aligned}
c_l^{(k+1)} &= c_l^{(k)} \\
&\qquad + \sum_{i=1}^{n} \gamma_i x_l^{(i)} (-s_i B_1^{(k)} x^{(i)} + \phi(0) - \phi(A_1^{(k)} x^{(i)})) \\
c_j^{(k+1)} &= c_j^{(k)} \\
&\qquad + \sum_{i=1}^{n} \gamma_i x_j^{(i)} (-s_i B_1^{(k)} x^{(i)} + \phi(0) - \phi(A_1^{(k)} x^{(i)})).
\end{aligned}$$

To simplify the notation, let

$$G_i^{(k)} = -s_i B_1^{(k)} x^{(i)} + \phi(0) - \phi(A_1^{(k)} x^{(i)}),$$

and thus

$$\begin{aligned}
\frac{c_l^{(k+1)}}{c_j^{(k+1)}} &= \frac{c_l^{(k)} + \sum_{i=1}^{n} \gamma_i x_l^{(i)} G_i^{(k)}}{c_j^{(k)} + \sum_{i=1}^{n} \gamma_i x_j^{(i)} G_i^{(k)}} \\
&= \frac{\frac{c_l^{(k)}}{c_j^{(k)} \prod_{p=1}^{d} x_j^{(p)}} + \sum_{i=1}^{n} \gamma_i \frac{x_l^{(i)}}{x_j^{(i)} c_j^{(k)} \prod_{p \neq i} x_j^{(p)}} G_i^{(k)}}{\frac{1}{\prod_{p=1}^{d} x_j^{(p)}} + \sum_{i=1}^{n} \gamma_i \frac{1}{c_j^{(k)} \prod_{p \neq i} x_j^{(p)}} G_i^{(k)}} \\
&\leq \frac{\frac{L}{\prod_{p=1}^{d} x_j^{(p)}} + \sum_{i=1}^{n} \gamma_i \frac{L}{c_j^{(k)} \prod_{p \neq i} x_j^{(p)}} G_i^{(k)}}{\frac{1}{\prod_{p=1}^{d} x_j^{(p)}} + \sum_{i=1}^{n} \gamma_i \frac{1}{c_j^{(k)} \prod_{p \neq i} x_j^{(p)}} G_i^{(k)}} \\
&= L
\end{aligned}$$

This completes the induction argument and as a consequence we obtain $c_l^{(t)} > 0$ and $\frac{c_l^{(t)}}{c_j^{(t)}} \leq L$ for all integers $t \geq 2$ and for $1 \leq l, j \leq d$.

Hence, the sequence $b_i^{(t)}$ is an upper bound for $a_i^{(t)}$ given learning rate $\gamma \leq \frac{1}{nLd}$. By symmetry between the rows of A, we have that, the solution given by solving the system $Bx^{(i)} = \phi^{-1}(x^{(i)})$ for $1 \leq i \leq n$ using gradient descent with constant learning rate is an entry-wise upper bound for the solution given by solving $\phi(Ax^{(i)}) = x^{(i)}$ for $1 \leq i \leq n$ using gradient descent with adaptive learning rate per training example when $A^{(0)} = B^{(0)} = \mathbf{0}$.

Now, since the entries of $B^{(t)}$ are bounded and since they are greater than the entries of $A^{(t)}$ for the given learning rate, it follows from the gradient update equation for A that the sequence of entries of $A^{(t)}$ are monotonically increasing from 0. Hence, if we show that the entries of $B^{(\infty)}$ are least upper bounds on the entries of $A^{(t)}$, then it follows that the entries of $A^{(t)}$ converge to the entries of $B^{(\infty)}$.

Suppose for the sake of contradiction that the least upper bound on the sequence $a_j^{(t)}$ (the j^{th} entry of the first row of A) is a $b_j^{(\infty)} - \epsilon_j$ for $\epsilon = [\epsilon_1, \dots \epsilon_d]$ with $\epsilon_j > 0$ for $1 \leq j \leq d$. Then

$$\phi(A_1^{(\infty)} x^{(i)}) = \phi(B_1^{(\infty)} x^{(i)} - \epsilon x^{(i)})$$

for $1 \leq i \leq n$. Since we are in the overparameterized setting, at convergence $A_1^{(\infty)}$ must give 0 loss under the mean squared error loss and so $\phi(B_1^{(\infty)}x^{(i)} - \epsilon x^{(i)}) = x_1^{(i)}$. This implies that $B_1^{(\infty)}x^{(i)} - \epsilon x^{(i)}$ is a pre-image of $x_1^{(i)}$ under ϕ. However, we know that $B_1^{(\infty)}x^{(i)}$ must be the minimum norm pre-image of $x_1^{(i)}$ under ϕ. Hence we reach a contradiction to minimiality since $B_1^{(\infty)}x^{(i)} - \epsilon x^{(i)} < B_1^{(\infty)}x^{(i)}$ as $\epsilon x^{(i)} > 0$. This completes the proof and so we conclude that $A^(t)$ converges to the solution given by autoencoding the linear system $Ax^{(i)} = \phi^{-1}x^{(i)}$ for $1 \leq i \leq n$ using gradient descent with constant learning rate. □

C. Single Layer Single Filter Convolutional Autoencoder

In the following, we present the proof for Theorem 3 from the main text.

Proof. A single convolutional filter with kernel size k and $\frac{k-1}{2}$ zero padding operating on an image of size $s \times s$ can be equivalently written as a matrix operating on a vectorized zero padded image of size $(s + k - 1)^2$. Namely, if $C_1, C_2, \ldots C_{k^2}$ are the parameters of the convolutional filter, then the layer can be written as the matrix

$$\begin{bmatrix} R \\ R_{r:1} \\ \vdots \\ R_{r:s-1} \\ R_{r:(s+k-1)} \\ \vdots \\ R_{r:(2s+k-2)} \\ \vdots \\ R_{r:((s+k-1)(s-1))} \\ \vdots \\ R_{r:((s+k)(s-1)} \end{bmatrix},$$

where

$$R = \begin{bmatrix} C_1 \ldots C_k \, \mathbf{0}_{s-1} \ldots C_{k^2-k+1} \ldots C_{k^2} \, \mathbf{0}_{(s+k)(s-1)} \end{bmatrix}$$

and $R_{r:t}$ denotes a right rotation of R by t elements.

Now, training the convolutional layer to autoencode example x using gradient descent is equivalent to training R to fit s^2 examples using gradient descent. Namely, R must satisfy $Rx = x_1, Rx_{l:1} = x_2, \ldots Rx_{l:(s+k-1)(s-1)+s-1} = x_{s^2}$ where $x_{l:t}^T$ denotes a left rotation of x^T by t elements. As in the proof for Theorem 1, we can use the general form of the solution for linear regression using gradient descent from Appendix A to conclude that the rank of the resulting solution will be s^2. □

D. Linearizing CNNs

In this section, we present how to extract a matrix form for convolutional and nearest neighbor upsampling layers. We first present how to construct a block of this matrix for a single filter in Algorithm 1. To construct a matrix for multiple filters, one need only apply the provided algorithm to construct separate matrix blocks for each filter and then concatenate them. We first provide an example of how to convert a single layer convolutional network with a single filter of kernel size 3 into a single matrix for 3×3 images.

First suppose we have a 3×3 image x as input, which is shown vectorized below:

$$\begin{bmatrix} x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 \\ x_7 & x_8 & x_9 \end{bmatrix} \rightarrow$$

$$\begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 & x_9 & \end{bmatrix}^T$$

Next, let the parameters below denote the filter of kernel size 3 that will be used to autoencode the above example:

$$\begin{bmatrix} A_1 & A_2 & A_3 \\ A_4 & A_5 & A_6 \\ A_7 & A_8 & A_9 \end{bmatrix}.$$

We now present the matrix form A for this convolutional filter such that A multiplied with the vectorized version of x will be equivalent to applying the convolutional filter above to the image x (the general algorithm to perform this construction is presented in Algorithm 1).

$$\begin{bmatrix} A_5 & A_6 & 0 & A_8 & A_9 & 0 & 0 & 0 & 0 \\ A_4 & A_5 & A_6 & A_7 & A_8 & A_9 & 0 & 0 & 0 \\ 0 & A_4 & A_5 & 0 & A_7 & A_8 & 0 & 0 & 0 \\ A_2 & A_3 & 0 & A_5 & A_6 & 0 & A_8 & A_9 & 0 \\ A_1 & A_2 & A_3 & A_4 & A_5 & A_6 & A_7 & A_8 & A_9 \\ 0 & A_1 & A_2 & 0 & A_4 & A_5 & 0 & A_7 & A_8 \\ 0 & 0 & 0 & A_2 & A_3 & 0 & A_5 & A_6 & 0 \\ 0 & 0 & 0 & A_1 & A_2 & A_3 & A_4 & A_5 & A_6 \\ 0 & 0 & 0 & 0 & A_1 & A_2 & 0 & A_4 & A_5 \end{bmatrix}$$

Importantly, this example demonstrates that the matrix corresponding to a convolutional layer has a fixed zero pattern. It is this forced zero pattern we use to prove that depth is required for memorization in convolutional autoencoders.

In downsampling autoencoders, we will also need to linearize the nearest neighbor upsampling operation. We provide the general algorithm to do this in Algorithm 2. Here, we provide a simple example for an upsampling layer with scale factor 2 operating on a vectorized zero padded 1×1 image:

Image Size	# Train. Ex.	Heuristic Lower Bound Layers	# of Layers	# of Filters Per Layer	# of Params	Spectrum
3×3	2	9	2	1	27	$1, 1, .98, \dots$
3×3	2	9	2	16	2592	$1, 1, .99, \dots$
3×3	2	9	2	128	149760	$1, 1, .99, \dots$
3×3	2	9	5	1	45	$1, 1, .78, \dots$
3×3	2	9	5	16	7200	$1, 1, .72, \dots$
3×3	2	9	5	128	444672	$1, 1, .7, \dots$
3×3	2	9	8	1	72	$1, 1, .14, \dots$
3×3	2	9	8	16	14112	$1, 1, .1, \dots$
3×3	2	9	8	128	887040	$1, 1, .08, \dots$

Table 1. Linear convolutional autoencoders with a varying number of layers and filters were initialized close to zero and trained on 2 normally distributed images of size 3×3. Memorization does not occur in any of the examples (memorization would be indicated by the spectrum containing two eigenvalues that are 1 and the remaining eigenvalues being close to 0). Increasing the number of filters per layer has minimal effect on the spectrum.

$$\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

The resulting output is a zero padded upsampled version of the input.

E. Deep Linear Convolutional Autoencoders Memorize

While Theorem 3 provided a lower bound on the depth required for memorization, Table 1 shows that the depth predicted by this bound is not sufficient. In each experiment, we trained a linear convolutional autoencoder to encode 2 randomly sampled images of size 3×3 with a varying number of layers and filters per layer. The first 3 rows of Table 1 show that the lower bound from Theorem 3 is not sufficient for memorization (regardless of overparameterization through filters) since memorization would be indicated by a rank 2 solution (with the third eigenvalue close to zero). In fact, the remaining rows of Table 1 show that even 8 layers are not sufficient for memorizing two images of size 3×3.

Image Size	# of Train. Ex.	Heuristic Lower Bound Layers	# of Layers	# of Params	Spectrum
2×2	1	2	3	27	$1, [< 10^{-2}]$
3×3	1	9	9	81	$1, [< 10^{-2}]$
4×4	1	29	29	261	$1, [< 10^{-3}]$
5×5	1	70	70	630	$1, [< 10^{-5}]$
6×6	1	144	144*	1296	$1, [< 2 \cdot 10^{-2}]$
7×7	1	267	267*	2403	$1, [< 4 \cdot 10^{-3}]$
3×3	3	9	10	90	$1, .98, .98, [< 10^{-2}]$
5×5	5	70	200*	1800	$1, 1, 1, 1, 1, [< 10^{-3}]$
7×7	5	144	350*	3105	$1, 1, 1, 1, 1, [< 10^{-2}]$

Table 2. Linear convolutional autoencoders with $\lceil \frac{s^4}{9} \rceil$ layers (as predicted by our heuristic lower bound) with a single filter per layer, initialized with each parameter as close to zero as possible, memorize training examples of size $s \times s$ similar to a single layer fully connected system.The bracket notation in the spectrum indicates that the magnitude of the remaining eigenvalues in the spectrum is below the value in the brackets.

Next we provide a heuristic bound to determine the depth needed to observe memorization (denoted by "Heuristic Lower Bound Layers" in Tables 1 and 2). Theorem 3 and Table 1 suggest that the number of filters per layer does not have an effect on the rank of the learned solution. We thus only consider networks with a single filter per layer with kernel size 3. It follows from Section 2 that overparameterized single layer fully connected autoencoders memorize training examples when initialized at 0. Hence, we can obtain a heuristic bound on the depth needed to observe memorization in linear convolutional autoencoders with a single filter per layer based on the number of layers needed for the network to have as many parameters as a fully connected network. The number of parameters in a single layer fully connected linear network operating on vectorized images of size $s \times s$ is s^4. Hence, using a single filter per layer with kernel size 3, the network needs $\lceil \frac{s^4}{9} \rceil$ layers to achieve the same number of parameters as a fully connected network. This leads to a heuristic lower bound of $\lceil \frac{s^4}{9} \rceil$ layers for memorization in linear convolutional autoencoders operating on images of size $s \times s$.

In Table 2, we investigate the memorization properties of networks that are initialized with parameters as close to zero as possible with the number of layers given by our heuristic lower bound and one filter of kernel size 3 per layer. The first 6 rows of the table show that all networks satisfying our heuristic lower bound have memorized a single training example since the spectrum consists of a single eigenvalue that is 1 and remaining eigenvalues with magnitude less than $\approx 10^{-2}$. Similarly, the spectra in the last 3 rows indicate that networks satisfying our heuristic lower bound also memorize multiple training examples, thereby suggesting that our bound is relevant in practice.

The experimental setup was as follows: All networks were trained using gradient descent with a learning rate of 10^{-1},

Algorithm 1 Create Matrix for Single Convolutional Filter given Input with dimensions $f \times s \times s$

Input: $parameters$:= parameters of f trained 3×3 CNN filters, s:= width and height of image without zero padding, f:= depth of image, $stride$:= stride of CNN filter

Output: Matrix C representing convolution operation

```
function CreateFilterMatrix($parameters, s, f, stride$)
    $paddedSize \leftarrow s + 2$
    $resized \leftarrow s/stride$
    $rowBlocks \leftarrow$ zeros matrix size $(f, (paddedSize)^2)$
    for $filterIndex \leftarrow 0$ to $f - 1$ do
        for $kernelIndex \leftarrow 0$ to $8$ do
            $rowIndex \leftarrow kernelIndex \mod 3 + paddedSize\lfloor \frac{kernelIndex}{3} \rfloor$
            $rowBlocks[filterIndex][rowIndex] \leftarrow parameters[filterIndex][kernelIndex]$
        end for
    end for
    $C \leftarrow$ zeros matrix of size $((resized + 2)^2, f \cdot paddedSize^2)$
    $index \leftarrow resized + 2 + 1$
    for $shift \leftarrow 0$ to $resized - 1$ do
        $nextBlock \leftarrow$ zeros matrix of size $(resized, f \cdot paddedSize^2)$
        $nextBlock[0] \leftarrow rowBlocks$
        for $rowShift \leftarrow 1$ to $resized - 1$ do
            $nextBlock[rowShift] \leftarrow rowBlocks$ shifted right by $stride \cdot rowShift$
        end for
        $C[index : index + resized, :] \leftarrow nextBlock$
        $index \leftarrow index + resize + 2$
        $rowBlock \leftarrow$ zero shift $rowBlock$ by $paddedSize \cdot stride$
    end for
    return $C$
end function
```

until the loss became less than 10^{-6} (to speed up training, we used Adam (Kingma & Ba, 2015) with a learning rate of 10^{-4} when the depth of the network was greater than 10). For large networks with over 100 layers (indicated by an asterisk in Table 2), we used skip connections between every 10 layers, as explained in (He et al., 2016), to ensure that the gradients can propagate to earlier layers. Table 2 shows the resulting spectrum for each experiment, where the eigenvalues were sorted by there magnitudes. The bracket notation indicates that all the remaining eigenvalues have magnitude less than the value provided in the brackets. Interestingly, our heuristic lower bound also seems to work for deep networks that have skip connections, which are commonly used in practice.

The experiments in Table 2 indicate that over 200 layers are needed for memorization of 7×7 images. In the next section, we discuss how downsampling can be used to construct much smaller convolutional autoencoders that memorize training examples.

F. Role of Downsampling for Memorization in Convolutional Autoencoders

To gain intuition for why downsampling can trade off depth to achieve memorization, consider a convolutional autoencoder that downsamples input to 1×1 representations through non-unit strides. Such extreme downsampling makes a convolutional autoencoder equivalent to a fully connected network; hence given the results in Section 2 such downsampling convolutional networks are expected to memorize. This is illustrated in Figure 3: The network uses strides of size 2 to progressively downsample to a 1×1 representation of a CIFAR10 input image. Training the network on two images from CIFAR10, the rank of the learned solution is exactly 2 with the top eigenvalues being 1 and the corresponding eigenvectors being linear combinations of the training images. In this case, using the default PyTorch initialization was sufficient in forcing each parameter to be close to zero.

Memorization using convolutional autoencoders is also observed with less extreme forms of downsampling. In fact, we observed that downsampling to a smaller representation and then operating on the downsampled representation

Algorithm 2 Create Matrix for Nearest Neighbor Upsampling Layer

Input: s:= width and height of image without zero padding, f:= depth of image, $scale$:= re-scaling factor for incoming image

Output: Matrix U representing convolution operation

```
function CreateUpsamplingMatrix($s, f, scale$)
    $outputSize \leftarrow s \cdot scale + 2$
    $U \leftarrow$ zeros matrix of size $(f \cdot outputSize^2, f \cdot (s+2)^2)$
    $index \leftarrow outputSize + 1$
    for $filterIndex \leftarrow 0$ to $f - 1$ do
        for $rowIndex \leftarrow 1$ to $s$ do
            for $scaleIndex \leftarrow 0$ to $scale - 1$ do
                for $columnIndex \leftarrow 0$ to $s$ do
                    $row \leftarrow$ zeros vector of size $(f(s+2)^2)$
                    $row[columnIndex + rowIndex(s+2) + filterIndex(s+2)^2] \leftarrow 1$
                    for $repeatIndex \leftarrow 0$ to $scale - 1$ do
                        $U[index] \leftarrow row$
                        $index \leftarrow index + 1$
                    end for
                end for
                $index \leftarrow index + 2$
            end for
        end for
        $index \leftarrow index + 2 \cdot outputSize$
    end for
    return $U$
end function
```

with depth provided by our heuristic bound established in Section E also leads to memorization. As an example, consider the network in Figure 4a operating on images from CIFAR10 (size 32×32). This network downsamples a 32×32 CIFAR10 image to a 4×4 representation after layer 1. As suggested by our heuristic lower bound for 4×4 images (see Table 2) we use 29 layers in the network. Figure 4b indicates that this network indeed memorized the image by producing a solution of rank 1 with eigenvalue 1 and corresponding eigenvector being the dog image.

G. Strong Memorization in Nonlinear Non-downsampling Autoencoders

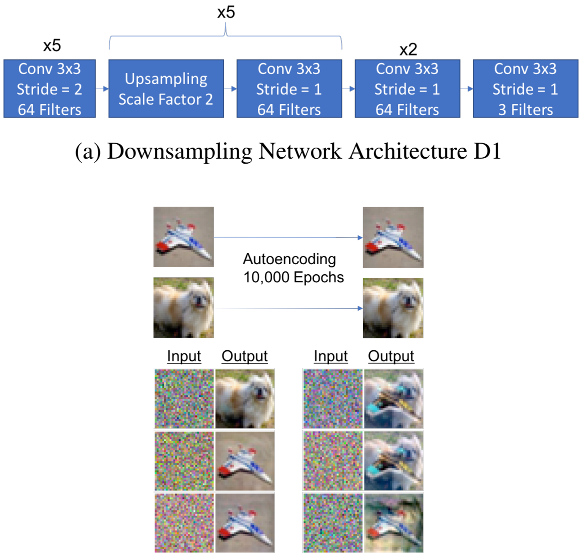
We start by investigating whether the heuristic bound on depth needed for memorization that we have established for linear convolutional autoencoders carries over to nonlinear convolutional autoencoders.

Example. *Consider a deep nonlinear convolutional autoencoder with a single filter per layer of kernel size* 3, 1 *unit of zero padding, and stride* 1 *followed by a leaky ReLU (Xu et al., 2015) activation that is initialized with parameters as close to 0 as possible. In Table 2 we reported that its linear counterpart memorizes* 4×4 *images with 29 layers. Figure 5 shows that also the corresponding nonlinear network with 29 layers can memorize* 4×4 *images. While the spectrum can be used to prove memorization in the linear setting, since we are unable to extract a nonlinear equivalent of the spectrum for these networks, we can only provide evidence for memorization by visual inspection.*

This example suggests that our results on depth required for memorization in deep linear convolutional autoencoders carry over to the nonlinear setting. In fact, when training on multiple examples, we observe that memorization is of a stronger form in the nonlinear case. Consider the example in Figure 6. We see that given new test examples, a nonlinear convolutional autoencoder with 5 layers trained on 2×2 images outputs *individual* training examples instead of combinations of training examples.

H. Robustness of Memorization and Role of Initialization

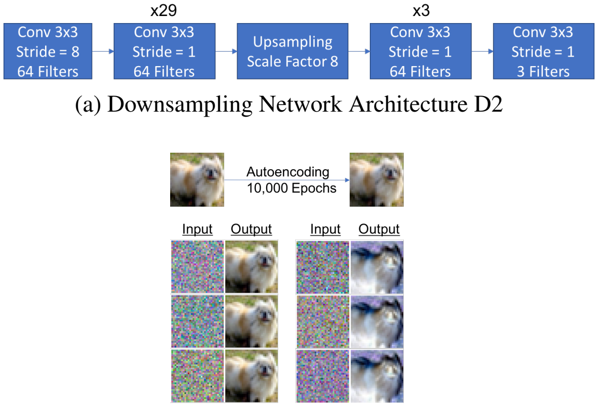
Memorization with Early Stopping. In all examples discussed so far, we trained the autoencoders to achieve nearly 0 error (less than 10^{-6}). In this section, we provide empirical evidence suggesting that the phenomenon of memorization is robust in the sense of appearing early in train-



(a) Downsampling Network Architecture D1

(b) Network D1 trained on a CIFAR10 Dog.

Figure 3. Downsampling Network Architecture D1 trained on a two images from CIFAR10 for 10000 iterations. Each parameter of the network is initialized using the default PyTorch initialization. When fed standard Gaussian data the model outputs a linear combination of training images since the corresponding solution has spectrum $1, 1, [< 2 \cdot 10^{-2}]$.



(a) Downsampling Network Architecture D2

(b) Network D2 trained on a CIFAR10 Dog.

Figure 4. Downsampling Network Architecture D2 trained on a single image from CIFAR10 for 10000 iterations. Each parameter of the network is initialized at $1.8 \cdot 10^{-3}$. When fed standard Gaussian data the model outputs a multiple of the training image since the corresponding solution has spectrum $1, [< 7 \cdot 10^{-6}]$.

ing, well before full convergence. The examples in Figure 7 using the network architecture shown in Figure 3a (where the nonlinear version is created by adding Leaky ReLU activation after every convolutional layer) illustrate this phenomenon. Both linear and nonlinear convolutional networks (that satisfy the heuristic conditions for memorization discussed in Sections 3 and F) show memorization throughout the training process.

As illustrated in Figure 7, networks in which training was terminated early, map a new given input to the current representation of the training examples. As shown in Figures 7a and 7b, the nonlinear autoencoder trained to autoencode two images from CIFAR10 clearly outputs the learned representations when given arbitrary test examples. As shown in Figures 7c and 7d, memorization is evident throughout training also in the linear setting, although the outputs are noisier than in the nonlinear setting.

Initialization at Zero is Necessary for Memorization. Section 2 showed that linear fully connected autoencoders initialized at zero memorize training examples by learning the minimum norm solution. Since in the linear setting the distance to the span of the training examples remains constant when minimizing the autoencoder loss regardless of the gradient descent algorithm used, non-zero initialization does not result in memorization. Hence, to see memorization, we require that each parameter of an autoencoder be

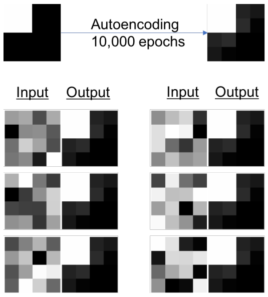


Figure 5. A 29 layer network with a single filter of kernel size 3, 1 unit of zero padding, and stride 1 followed by a leaky ReLU activation per layer initialized with every parameter set to 10^{-1} memorizes 4×4 images. Our training image consists of a white square in the upper left hand corner and the test examples contain pixels drawn from a standard normal distribution.

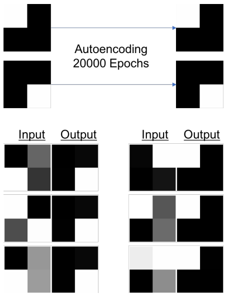
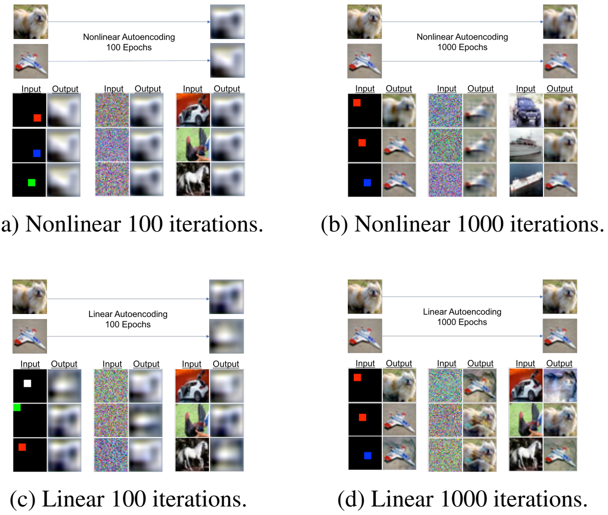


Figure 6. A 5 layer nonlinear network strongly memorizes 2×2 images. The network has a single filter of kernel size 3, 1 unit of zero padding, and stride 1 followed by a leaky ReLU activation per layer with Xavier Uniform initialization. The network also has skip connections between every 2 layers. The training images are orthogonal: one with a white square in the upper left corner and one with a white square in the lower right corner. The test examples contain pixels drawn from a standard normal distribution.



(a) Nonlinear 100 iterations.

(b) Nonlinear 1000 iterations.

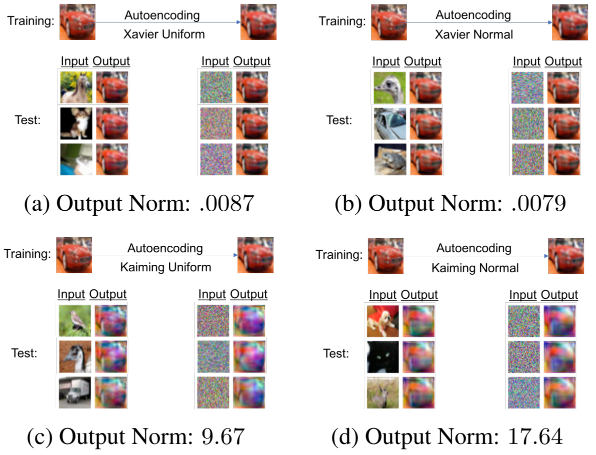
(c) Linear 100 iterations.

(d) Linear 1000 iterations.

Figure 7. The linear and nonlinear version of the network from Figure 3a both memorize learned representations of training data throughout the training process.

initialized as close to zero as possible (while allowing for training).

We now briefly discuss how popular initialization techniques such as Kaiming uniform/normal (He et al., 2015), Xavier uniform/normal (Glorot & Bengio, 2010), and default PyTorch initialization (Paszke et al., 2017) relate to zero initialization. In general, we observe that Kaiming uniform/normal initialization leads to an output with a larger ℓ_2 norm as compared to a network initialized using Xavier uniform/normal or PyTorch initializations. Thus, we do not expect Kaiming uniform/normal initialized networks to present memorization as clearly as the other initialization schemes. That is, for linear convolutional autoencoders, we expect these networks to converge to a solution further from the minimum nuclear norm solution and for nonlinear convolutional autoencoders, we expect these networks to produce noisy versions of the training examples when fed arbitrary inputs. This phenomenon is demonstrated experimentally in the examples in Figure 8.



(a) Output Norm: .0087

(b) Output Norm: .0079

(c) Output Norm: 9.67

(d) Output Norm: 17.64

Figure 8. Effect of popular initialization strategies on memorization: Each figure demonstrates how the nonlinear version of the autoencoder from Figure 3a (modified with Leaky ReLU activations after each convolutional layer) behaves when initialized using Xavier uniform/normal and Kaiming uniform/normal strategies. We also give the ℓ_2 norm of the output for the training example prior to training. Consistent with our predictions, the Kaiming uniform/normal strategies have larger norms and the output for arbitrary inputs shows that memorization is noisy.