Identity Crisis: Memorization and Generalization Under Extreme Overparameterization

Chiyuan Zhang[1], Samy Bengio[1], Moritz Hardt[2], Michael C. Mozer[1,3], and Yoram Singer[1,4]

[1] *Google Brain*
[2] *University of California, Berkeley*
[3] *University of Colorado, Boulder*
[4] *Princeton University*

Abstract

We study the interplay between memorization and generalization of overparametrized networks in the extreme case of a single training example. The learning task is to predict an output which is as similar as possible to the input. We examine both fully-connected and convolutional networks that are initialized randomly and then trained to minimize the reconstruction error. The trained networks take one of the two forms: the constant function ("memorization") and the identity function ("generalization"). We show that different architectures exhibit vastly different inductive bias towards memorization and generalization.

1 Introduction

The empirical success of deep networks is often attributed to large data sets for training. However, sample size does not provide a comprehensive rationale since complex models often outperform simple models on a given data set, even when the number of free parameters of each far exceeds the number of training examples. What form of inductive bias could achieve state-of-the-art performance from highly overparameterized models? Numerous theoretical and empirical studies of inductive bias in deep learning have been conducted in recent years [12, 16, 5, 23, 20, 24, 3, 31] but these postmortem analyses do not identify the root source of the bias.

One cult belief among researchers is that gradient-based optimization methods provide an implicit bias toward simple solutions. However, when a network is sufficiently large, gradient methods provably fit the training set [2, 9, 10, 32]. These results do not distinguish a model trained on a data distribution with strong statistical regularities from one trained on randomly shuffled labels. Although the former model might achieve good generalization, the latter can only memorize the training labels. Consequently, these analyses do not shed much light on the question of inductive bias. Another line of research characterizes sufficient conditions on the data distribution that guarantee generalization from a trained network [8, 19]. This direction, while very promising, has thus far identified only structures that can be solved by linear or nearest neighbor classifiers operating in the original input space. The fact that in many applications deep neural networks significantly outperform these simpler models reveals a gap in our understanding.

In this article, we conduct a novel empirical exploration in a highly restrictive setting that admits visualization and quantification of inductive bias, allowing us to compare variations in architecture, network depth, optimization procedure, initialization scheme, and hyperparameters. The particular task we investigate is learning an *identity mapping* in a regression setting. The identity mapping is interesting for several reasons. First, it imposes a structural regularity between the input and output, the type of regularity that could in principle lead to systematic generalization. Second, it requires that every input feature is transmitted to the output and thus provides a sensitive indicator of whether a model succeeds in passing activations (and gradients) between inputs and outputs. Third, conditional image generation is a popular task in the literature [e.g., 21, 18]; an identity mapping is the simplest form of such a generative process. Fourth, and perhaps most importantly, it affords analysis and visualization of model behavior.

Consider networks trained on the identity task with 60k MNIST digits. Although only digit images are presented during the training, one might expect the strong regularity of the task to lead to good generalization to images other

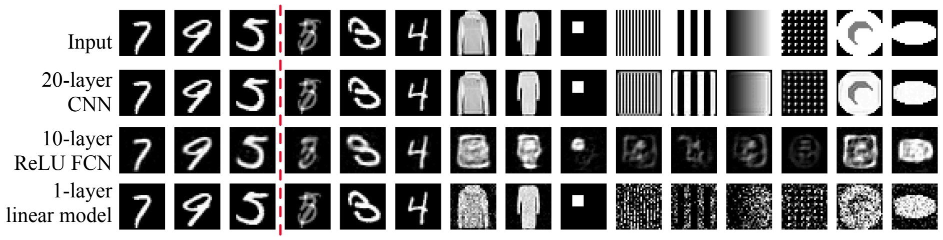


Figure 1: **Predictions of three architectures trained on identity task with 60k MNIST examples.** The red dashed line separates 3 training examples from the test examples.

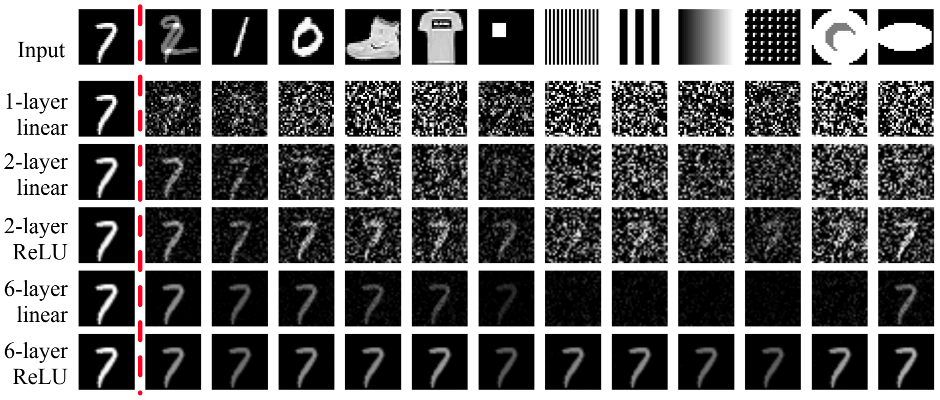


Figure 2: **Visualization of the predictions from fully connected networks trained on a single example.** The first row shows inputs that consist of the (single) training digit (1st column) and unseen test examples.

than digits. Figure 1 compares a 20-layer ConvNet, a 10-layer fully connected net with ReLU activation, and a 1-layer fully-connected net. All nets perform well on the training set (first three columns) and transfer well to novel digits and digit blends (columns 4–6), but outside of the manifold of hand-printed digits, only the ConvNet discovers a reasonably good approximation to the identity function. Figure 1 reflects architecture-specific inductive bias that persists even with 60k training examples. Despite this persistence, a model's intrinsic bias is more likely to be revealed with a smaller training set. In this article, we push this argument to the limit by studying learning with a *single* training example. Our initial intuition was that a single example would be uninteresting because models would simply induce a constant output (e.g., via biases on output units). Further, it seemed inconceivable that biases would be sufficiently strong to learn a mapping close to the identity. Unexpectedly, our experiments show that model behavior is subtle and architecture dependent. In a broad set of experiments, we highlight model features—including depth, initialization, and hyperparameters—that determine where a model lands on the continuum between *memorization* (i.e., learning a constant function) and *generalization* (i.e., learning the identity function). The simplicity of the training scenario permits this quantitative characterization of inductive bias.

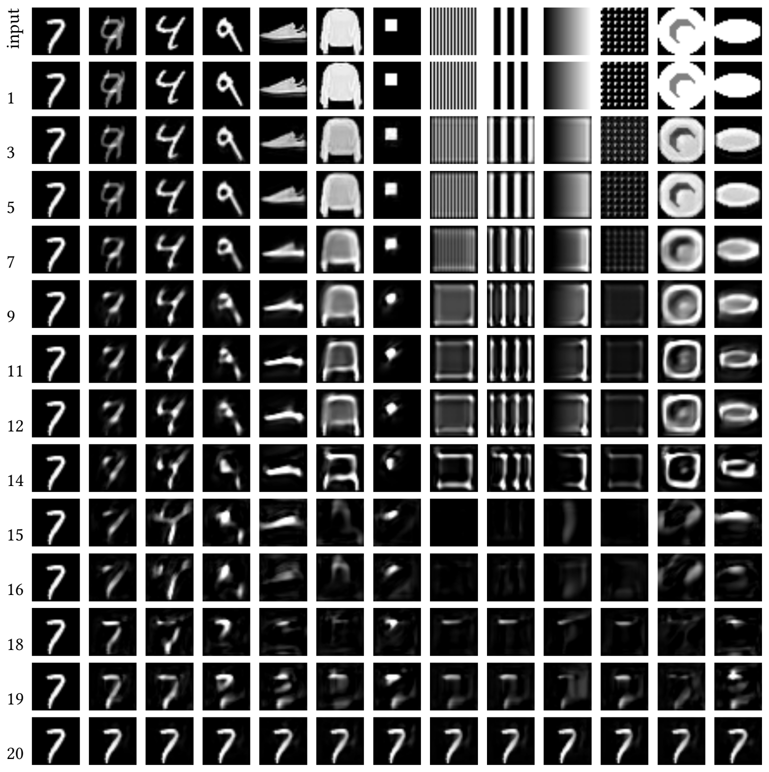
2 Learning the identity function under extreme overparameterization

The simplest neural network is a one layer fully connected linear model, $f_W(x) = Wx$, where $x \in \mathbb{R}^d$ and $W \in \mathbb{R}^{d\times d}$. The optimization problem is convex and well understood. Due to overparameterization, no unique solution exists. However, given randomly initialized weights W^0, gradient descent converges to a unique global minimizer:

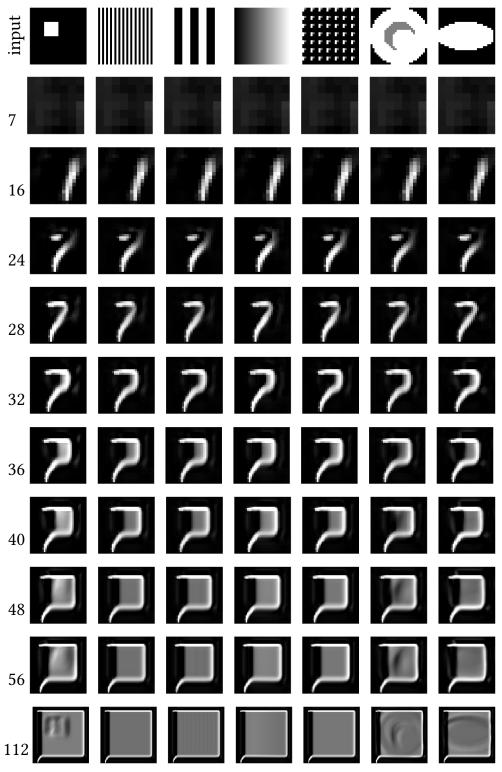
$$\hat{f}_W(x) = \frac{\hat{x}^\top x}{\|\hat{x}\|_2^2} \cdot \hat{x} + W^0 \left(x - \frac{\hat{x}^\top x}{\|\hat{x}\|_2^2} \cdot \hat{x} \right) \tag{1}$$

(see Appendix D). Consequently, the prediction of a model trained on $\hat{x}$ is fully determined: the test example x is decomposed into the component in the direction of $\hat{x}$ and its orthogonal component. The component parallel to $\hat{x}$ will make the prediction look like $\hat{x}$, whereas the orthogonal component entirely depends on the random initialization.

In this case, the learning algorithm has a strong inductive bias that yields a unique solution given W^0 despite the overparameterization. However, this inductive bias does not magically lead to model generalization: the trained model fails to learn the identity function. Although it predicts well when the input has a strong correlation with training example $\hat{x}$, otherwise predictions are random. In particular, when the test example x is orthogonal to $\hat{x}$, the prediction is completely random. The second row of Figure 2 shows an example of predictions from such a model.



(a) predictions of convNets with different depth



(b) different input sizes

Figure 3: **Visualization of predictions from ConvNets trained with one MNIST example.** (a) shows the predictions from ConvNets with different depths, indicated by the numbers the left; (b) shows the predictions from a 20-layer ConvNets, when evaluated on inputs with different sizes, indicated by the numbers on the left.

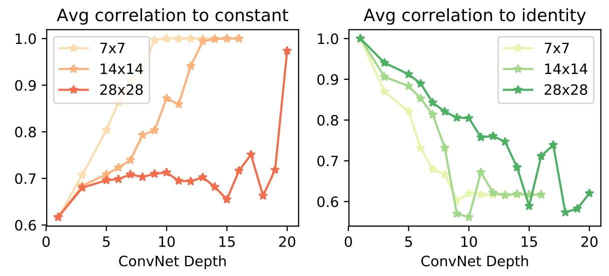
The deviations from the identity function are quite different in multilayer fully connected networks. A multilayer *linear* network with no bottleneck in the hidden space has the same representational power as a 1-layer linear network, but its learning dynamics are nonconvex and that alters the inductive biases. Figure 2 shows that a 6-layer linear network behaves more like a 6-layer ReLU network than like the 1-layer convex case, with a strong bias towards a constant function that maps every input to the single training image. See Appendix E for more details.

The situation is very different for ConvNets (Figure 3a). Compared to general fully connected FCNs, ConvNets have strong structural constraints that limit the receptive field of each neuron to a spatially local neighborhood and the same weights are re-used across the spatial map. The two constraints indeed match the structure of the identity target function. (Appendix C.3 explores other cases of using ConvNets to construct the identity function.)

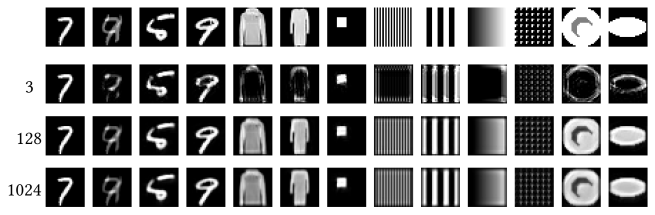
The Figure shows that, except for some artifacts on the boundaries, ConvNets with a depth of up-to-5 hidden layers learn a good approximation to the identity function. As the depth increases, the ConvNets bias towards the constant function. Visualization of intermediate representations and quantitative evaluations can be found in Appendix F.

3 Robustness of inductive biases

The bias towards either constant or identity map is not explicitly encoded in the loss function, but rather is implicit in the network architecture, the learning algorithm, and other hyperparameters. To assess the robustness of the inductive bias, we systematically manipulate critical factors in training and testing. We focus on ConvNets, due to their proven performance and the interpretability of their hidden layers. Figure 3b shows predictions of a 20-layer ConvNet that has learned the constant function on a 28×28 image and is tested on a set of inputs scaled to varying resolutions (indicated to the left of each row). The learned constant map holds up for input dimensions slightly smaller



(a) **Comparing bias towards constant and identity when trained with different image sizes.** The x-axis is the depth of the ConvNets, while the y-axis is the mean correlation to either the constant or the identity map. Each curve corresponds to training with a different image size.



(b) **Visualization of the predictions from 5-layer ConvNets with various number of hidden channels.** Numbers on the left indicate the number of channels. For the intermediate layers, the numbers indicate the number of both the input and the output channels. The number of input channels for the bottom layer and the output channels for the top layer are decided by the data (one for our case).

Figure 4: Evaluation of the robustness of inductive biases by varying different hyperparameters during training of ConvNets.

than the trained image. On the other hand, as the input dimension increases, it is interesting to see smooth changes which reveal the network's understanding of the "7". Because of the fully convolutional network structure, output units do not have individual biases at each spatial location. Consequently, the network must learn to synthesize the "7" through multiple layers of transformations, a nontrivial generative task. The Figure suggests that the network starts from the outside, where the artifacts due to the boundary padding in convolution provide spatial cues. The visualization of the intermediate layers in Figure 19 of Appendix H.1 confirms this hypothesis. See Appendix H for more details and other analysis.

We also explored the consequence of *training* on images of varying dimensions. Figure 4a shows the correlation of the model output to the constant and the identity functions for models trained on 7×7, 14×14, and 28×28 images. All three show more bias toward the constant function and less bias toward the identity function with increased network depth. However, smaller input grids accentuate the effect, likely because the training image has fewer pixels to constrain the mapping.

Figure 4b explores the effect of varying the number of channels. Two channels are sufficient to encode the identity function for grayscale inputs (Appendix C.3). However, the inductive bias of a trained model depends on the channel count. The aggressively over-parameterized network with 1024 channels (~25M parameters per middle-layer convolution module) does not seem to suffer from overfitting, whereas the network with three channels misses the center content of many predicted images. Underfitting is not an issue for the small network because it reconstructs the training image (first column) correctly. Our results could be explained by assuming that a model's behavior is very sensitive to the initialization of the channels. With fewer channels, it is far more likely that all channels are initialized poorly for conveying information (see Appendix H.3). Studies of different training initialization schemes indeed confirm that random initial conditions can have a big impact on the inductive bias (see Appendix H).

Conclusions

We presented empirical studies of the extreme case of overparameterization when learning from a single example. We investigated the interplay between memorization and generalization in deep neural networks. By restricting the learning task to the identity function, we sidestepped issues such as the underlying optimal Bayes error of the problem and the approximation error of the hypothesis classes. This choice also facilitated rich visualization and intuitive interpretation of the trained models. Under this setup, we investigate gradient-based learning procedures with explicit memorization-generalization characterization. Our results indicate that different architectures exhibit vastly different inductive bias towards memorization and generalization.

Acknowledgments We would like to thank Kunal Talwar, Hanie Sedghi, and Rong Ge for helpful discussions.

References

[1] Zeyuan Allen-Zhu, Yuanzhi Li, and Yingyu Liang. Learning and generalization in overparameterized neural networks, going beyond two layers. *CoRR*, 1811.04918, 2018.

[2] Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via Over-Parameterization. *CoRR*, arXiv:1811.03962, 2018.

[3] Sanjeev Arora, Rong Ge, Behnam Neyshabur, and Yi Zhang. Stronger generalization bounds for deep nets via a compression approach. *CoRR*, arXiv:1802.05296, 2018.

[4] Devansh Arpit, Stanisław Jastrzębski, Nicolas Ballas, David Krueger, Emmanuel Bengio, Maxinder S Kanwal, Tegan Maharaj, Asja Fischer, Aaron Courville, Yoshua Bengio, et al. A closer look at memorization in deep networks. In *International Conference on Machine Learning*, 2017.

[5] Peter L Bartlett, Dylan J Foster, and Matus J Telgarsky. Spectrally-normalized margin bounds for neural networks. In *Advances in Neural Information Processing Systems*, pages 6240–6249, 2017.

[6] Raef Bassily, Mikhail Belkin, and Siyuan Ma. On exponential convergence of SGD in non-convex over-parametrized learning. *CoRR*, arXiv:1811.02564, 2018.

[7] Mikhail Belkin, Daniel Hsu, and Partha Mitra. Overfitting or perfect fitting? risk bounds for classification and regression rules that interpolate. In *Advances in Neural Information Processing Systems*, 2018.

[8] Alon Brutzkus, Amir Globerson, Eran Malach, and Shai Shalev-Shwartz. SGD learns over-parameterized networks that provably generalize on linearly separable data. In *ICLR*, 2018.

[9] Simon S Du, Jason D Lee, Haochuan Li, Liwei Wang, and Xiyu Zhai. Gradient descent finds global minima of deep neural networks. *CoRR*, arXiv:1811.03804, 2018.

[10] Simon S Du, Xiyu Zhai, Barnabas Poczos, and Aarti Singh. Gradient descent provably optimizes over-parameterized neural networks. *CoRR*, arXiv:1810.02054, 2018.

[11] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[12] Gintare Karolina Dziugaite and Daniel M Roy. Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data. In *UAI*, 2016.

[13] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[14] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[16] Kenji Kawaguchi, Leslie Pack Kaelbling, and Yoshua Bengio. Generalization in deep learning. *CoRR*, arXiv:1710.05468, 2017.

[17] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[18] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4681–4690, 2017.

[19] Yuanzhi Li and Yingyu Liang. Learning overparameterized neural networks via stochastic gradient descent on structured data. In *NIPS*, 2018.

[20] Tengyuan Liang, Tomaso Poggio, Alexander Rakhlin, and James Stokes. Fisher-rao metric, geometry, and complexity of neural networks. *CoRR*, arXiv:1711.01530, 2017.

[21] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[22] Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. In search of the real inductive bias: On the role of implicit regularization in deep learning. *CoRR*, arXiv:1412.6614, 2014.

[23] Behnam Neyshabur, Srinadh Bhojanapalli, David McAllester, and Nati Srebro. Exploring generalization in deep learning. In *Advances in Neural Information Processing Systems*, pages 5947–5956, 2017.

[24] Behnam Neyshabur, Srinadh Bhojanapalli, and Nathan Srebro. A PAC-Bayesian approach to Spectrally-Normalized margin bounds for neural networks. In *ICLR*, 2018.

[25] Samet Oymak and Mahdi Soltanolkotabi. Overparameterized nonlinear learning: Gradient descent takes the shortest path? *CoRR*, arXiv:1812.10004, 2018.

[26] Adityanarayanan Radhakrishnan, Mikhail Belkin, and Caroline Uhler. Downsampling leads to image memorization in convolutional autoencoders. *CoRR*, arXiv:1810.10333, 2018.

[27] Vatsal Shah, Anastasios Kyrillidis, and Sujay Sanghavi. Minimum norm solutions do not always generalize well for over-parameterized problems. *CoRR*, arXiv:1811.07055, 2018.

[28] Daniel Soudry, Elad Hoffer, Mor Shpigel Nacson, Suriya Gunasekar, and Nathan Srebro. The implicit bias of gradient descent on separable data. *Journal of Machine Learning Research*, 19(70), 2018.

[29] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[30] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *International Conference on Learning Representations*, 2017.

[31] Wenda Zhou, Victor Veitch, Morgane Austern, Ryan P Adams, and Peter Orbanz. Non-vacuous generalization bounds at the ImageNet scale: a PAC-Bayesian compression approach. In *ICLR*, 2019.

[32] Difan Zou, Yuan Cao, Dongruo Zhou, and Quanquan Gu. Stochastic gradient descent optimizes over-parameterized deep ReLU networks. *CoRR*, arXiv:1811.08888, 2018.

A Related work

The consequences of overparameterized models in deep learning have been extensively studied in recently years, on the optimization landscape and convergence of SGD [2, 9, 10, 6, 32, 25], as well as the generalization guarantees under stronger structural assumptions of the data [19, 8, 1]. Another line of related work is the study of the implicit regularization effects of SGD on training overparameterized models [22, 30, 28, 27].

The traits of memorization in learning are also explicitly studied from various perspectives such as prioritizing learning of simple patterns [4] or perfect interpolation of the training set [7]. More recently, coincidently with the writing of this paper, Radhakrishnan et al. [26] reported on the effects of the downsampling operator in convolutional auto-encoders on image memorization. Their empirical framework is similar to ours, fitting ConvNets to the autoregression problem with few training examples. We focus on investigating the general inductive bias in the extreme overparameterization case, and study a broader range of network types without enforcing a bottleneck in the architectures.

B Experiment details and hyper-parameters

We specify the experiment setups and the hyper-parameters here. Unless otherwise specified in each study (e.g. when we explicitly vary the number of convolution channels), all the hyper-parameters are set according to the default values here.

The main study is done with the MNIST dataset. It consists of grayscale images of hand written digits of size 28×28. For training, we randomly sample one digit from the training set (a digit '7') with a fixed random seed. For testing, we use random images from the test set of MNIST and Fashion-MNIST, as well as algorithmically generated structured patterns and random images. The training and test images are all normalized by mapping the pixel values in $\{0, 1, \ldots, 255\}$ to $[0, 1]$, and then standardize with the mean 0.1307 and standard deviation 0.3081 originally calculated on the (full) MNIST training set.

The models are trained by minimizing the mean squared error (MSE) loss with a vanilla SGD (base learning rate 0.01 and momentum 0.9). The learning rate is scheduled as stagewise constant that decays with a factor of 0.2 at the 30%, 60% and 80% of the total training steps (2,000,000). No weight decay is applied during training.

For neural network architectures, the rectified linear unit (ReLU) activation is used for both fully connected networks (FCNs) and convolutional networks (ConvNets). The input and output dimensions are decided by the data. The hidden dimensions for the FCNs are 2,048 by default. The ConvNets use 5×5 kernels with stride 1 and padding 2, so that the geometry does not change after each convolution layer.

C Representation of the identity function using deep networks

In this section, we provide explicit constructions on how common types of neural networks can represent the identity function. Those constructions are only proof for that the models in our study have the capacity to represent the target function. There are many different ways to construct the identity map for each network architecture, but we try to provide the most straightforward and explicit constructions. However, during our experiments, even when the SGD learns (approximately) the identity function, there is no evidence suggesting that it is encoding the functions in similar ways as described here. We put some mild constraints (e.g. no "bottleneck" in the hidden dimensions) to allow more straightforward realization of the identity function, but this by no means asserts that networks violating those constraints cannot encode the identity function.

C.1 Linear models

For a one-layer linear network $f(x) = Wx$, where $W \in \mathbb{R}^{d \times d}$, setting W to the identity matrix will realize the identity function. For a multi-layer linear network $f(x) = (\prod_\ell W_\ell)x$, we need to require that all the hidden dimensions are not smaller than the input dimension. In this case, a simple concrete construction is to set each W_ℓ to an identity matrix.

C.2 Multi-layer ReLU networks

The ReLU activation function $\sigma(\cdot) = \max(0, \cdot)$ discards all the negative values. There are many ways one can encode the negative values and recover it after ReLU. We provide a simple approach that uses hidden dimensions twice the input dimension. Consider a ReLU network with one hidden layer $f(x) = W_2\sigma(W_1 x)$, where $W_2 \in \mathbb{R}^{d\times 2d}$, $W_1 \in \mathbb{R}^{2d\times d}$. The idea is to store the positive and negative part of x separately, and then re-construct. This can be achieved by setting

$$W_1 = \begin{pmatrix} I_d \\ -I_d \end{pmatrix}, \quad W_2 = \begin{pmatrix} I_d & -I_d \end{pmatrix}$$

where I_d is the d-dimensional identity matrix. For the case of more than two layers, we can use the bottom layer to split the positive and negative part, and the top layer to merge them back. All the intermediate layers can be set to $2d$-dimensional identity matrix. Since the bottom layer encode all the responsives in non-negative values, the ReLU in the middle layers will pass through.

C.3 Convolutional networks

In particular, we consider 2D convolutional networks for data with the structure of multi-channel images. A mini-batch of data is usually formatted as a four-dimensional tensor of the shape $B \times C \times H \times W$, where B is the batch size, C the number of channels (e.g. RGB or feature channels for intermediate layer representations), H and W are image height and width, respectively. A convolutional layer (ignoring the bias term) is parameterized with another four-dimensional tensor of the shape $\bar{C} \times C \times K_H \times K_W$, where $\bar{C}$ is the number of output feature channels, K_H and K_W are convolutional kernel height and width, respectively. The convolutional kernel is applied at local $K_H \times K_W$ patches of the input tensor, with optional padding and striding.

For one convolution layer to represent the identity function, we can use only the center slice of the kernel tensor and set all the other values to zero. Note it is very rare to use even numbers as kernel size, in which case the "center" of the kernel tensor is not well defined. When the kernel size is odd, we can set

$$W_{\bar{c}chw} = \begin{cases} 1 & \bar{c} = c,\ h = \lfloor K_H/2 \rfloor,\ w = \lfloor K_W/2 \rfloor \\ 0 & \text{otherwise} \end{cases}$$

By using only the center of the kernel, we essentially simulate a 1×1 convolution, and encode a local identity function for each (multi-channel) pixel.

For multi-layer convolutional networks with ReLU activation functions, the same idea as in multi-layer fully-connected networks can be applied. Specifically, we ask for twice as many channels as the input channels for the hidden layers. At the bottom layer, separately the positive and negative part of the inputs, and reconstruct them at the top layer.

D Closed form solution for single-layer overparameterized network

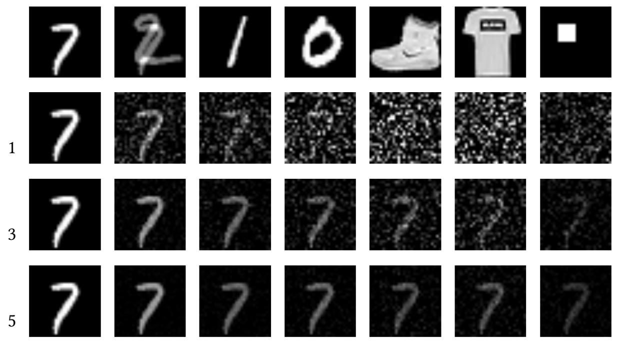
In (1), a closed form solution is provided for the global minimizer of a one-layer neural network trained on a single example. The derivation of the solution is presented here. Let $\hat{x}$ be the training example, the gradient of the empirical risk (the mean squared error on the single training example $\hat{x}$) is

$$\frac{\partial \hat{R}}{\partial W} = (W - I)\hat{x}\hat{x}^\top \tag{2}$$

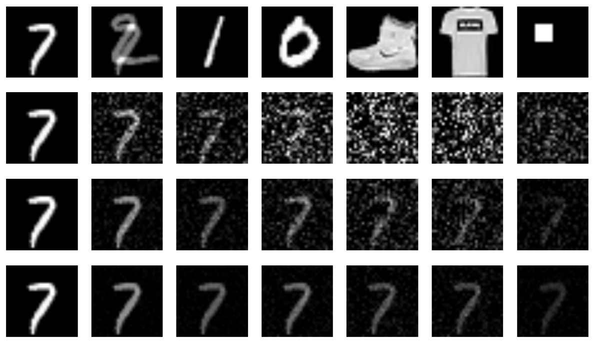
Gradient descent with step sizes η_t and initialization weights W^0 update weights as

$$\begin{aligned} W^T &= W^0 - \sum_{t=1}^{T} \eta_t (W^{t-1} - I)\hat{x}\hat{x}^\top \\ &:= W^0 + u_T \hat{x}^\top \end{aligned}$$

where $u_T \in \mathbb{R}^d$ is a vector decided via the accumulation in the optimization trajectory. Because of the form of the gradient, it is easy to see the solution found by gradient descent will always have such parameterization structure.



(a) Hidden dimension 784 (= input dimension)



(b) Hidden dimension 2048

Figure 5: **Visualization of predictions from trained multi-layer linear networks.** The first row shows the test images, and the remaining rows shows the prediction from a trained linear network with 1, 3, and 5 hidden layers, respectively.

Moreover, under this parameterization, a unique minimizer exists that solves the equation

$$\hat{x} = f_W(\hat{x}) = W^0\hat{x} + u\hat{x}^\top\hat{x}$$

via

$$\hat{u} = \frac{(I - W^0)\hat{x}}{\|\hat{x}\|_2^2} \tag{3}$$

Therefore, the global minimizer can be written as in (1), copied here for convenience:

$$\begin{aligned}\hat{f}_W(x) &= (W^0 + \hat{u}\hat{x}^\top)x \\ &= \frac{\hat{x}^\top x}{\|\hat{x}\|_2^2} \cdot \hat{x} + W^0\left(x - \frac{\hat{x}^\top x}{\|\hat{x}\|_2^2} \cdot \hat{x}\right)\end{aligned}$$

For the one-layer network case, the optimization problem is convex. Under standard conditions in convex optimization, gradient descent will converge to the global minimizer shown above.

The calculation can be easily extended to arbitrary target function other than the identity, as well as the case with multiple training examples. It will result in a decomposition of the test example into the subspace spanned by the training samples, and the orthogonal subspace. In particular, when the training examples have full rank — spanning the whole input space, the model will correctly learn the identity function.

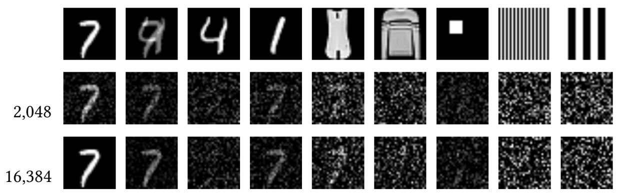
E Full results of fully connected multi-layer networks

E.1 Fully connected linear networks

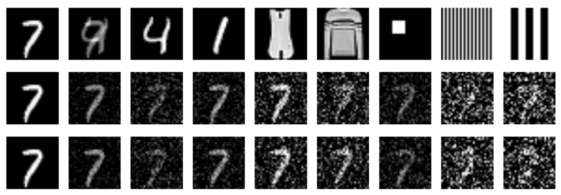
Figure 5 shows the results on multi-layer *linear* networks with various number of hidden layers and hidden units. The model with only one hidden layer resembles the convex case (see Figure 2), but as the depth increases, the model tends toward a constant function that maps every input to the single training image. The depth of the architecture has a stronger effect on the inductive bias than the width. For example, the network with one hidden layer of dimension 2048 has 3.2M parameters, more than the 2.5M parameters of the network with three hidden layers of dimension 784. But the latter behaves less like the convex case.

E.2 Two-layer fully connected ReLU networks

Li and Liang [19] offer a theoretical characterization of learning in a two-layer ReLU neural network. They show that when the data consist of well separated clusters (i.e., the cluster diameters are much smaller than the distances between each cluster pair), training an overparameterized two-layer ReLU network will generalize well. To simplify



(a) Training bottom layer only



(b) Training both layers

Figure 6: **Visualization of predictions from two-layer ReLU networks.** The first row shows the test images, and the remaining rows shows the predictions from trained models with hidden dimension 2,048 and 16,384, repectively.

the analysis, they study a special case where the weights in the top layer are randomly initialized and fixed; only the bottom layer weights are learned.

We study the problem of learning a two-layer ReLU network under our identity-mapping task. Figure 6 compares the cases of learning the bottom layer only and learning both layers (left and right panels, respectively). The two cases demonstrate different inductive biases for predictions on unseen test images. When only the first layer is trained, the tendency is toward speckled noise, but when both layers are trained, the tendency is toward a constant output (the image used for training). Our observation does not contradict the theoretical results of Li and Liang [19], which assume a well separated and clustered data distribution.

For bottom-layer only training, the analysis is similar to that for one-layer networks. Let us denote

$$f_W(x) = \langle \alpha, \mathrm{ReLU}(z) \rangle, \quad z = Wx \tag{4}$$

where $W \in \mathbb{R}^{m \times d}$ is the learnable weight matrix, and $\alpha \in \mathbb{R}^m$ is randomly initialized and fixed. Although the trained weights no longer have a closed-form solution, the solution found by gradient descent is always parameterized as

$$W^T = W^0 + u^T \hat{x}^\top \tag{5}$$

where $\hat{x}$ is the training example, and $u^T \in \mathbb{R}^m$ summarizes the efforts of gradient descent up to time T. In particular, the gradient of the empirical risk $\hat{R}$ with respect to each row $W_{:r}$ of the learnable weight is

$$\frac{\partial \hat{R}}{\partial W_{:r}} = \frac{\partial \hat{R}}{\partial z_r} \frac{\partial z_r}{\partial W_{:r}} = \frac{\partial \hat{R}}{\partial z_r} \hat{x}^\top \tag{6}$$

Putting it together, the full gradient is

$$\frac{\partial \hat{R}}{\partial W} = \frac{\partial \hat{R}}{\partial z} \cdot \hat{x}^\top \tag{7}$$

Since the gradient lives in the span of the training example $\hat{x}$, the solution found by gradient descent is always parameterized as (5).

The same arguments applies to multi-layer neural networks. The prediction on any test example that is orthogonal to $\hat{x}$ will depend only on randomly initialized W^0 and upper layer weights. When only the bottom layer is trained, the upper layer weights will also be independent from the data, therefore the prediction is completely random. However, when all the layers are jointly trained, the arguments no longer apply. The empirical results presented later in the paper that multi-layer networks bias towards the constant function verify this.

Consequently, if the test example is orthogonal to $\hat{x}$ (i.e., $\hat{x}^\top x = 0$), the prediction depends solely on the randomly initialized values in W^0 and therefore can be characterized by the distribution used for parameter initialization.

However, when both layers are trained, the upper layer weights are also tuned to make the prediction fit the training output. In particular, the learned weights in the upper layer depend on W^0. Therefore, the randomness arguments shown above no longer apply even for test examples orthogonal to $\hat{x}$. As the empirical results show, the behavior is indeed different.

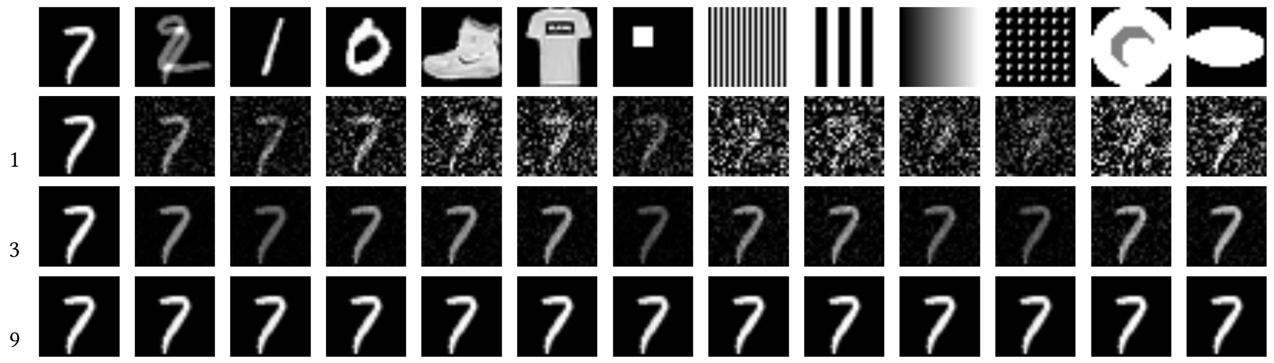
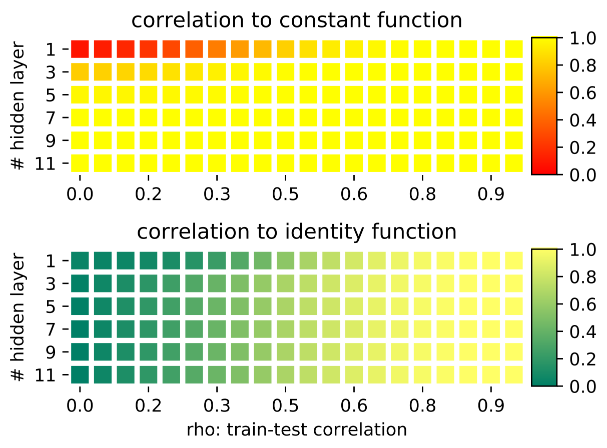
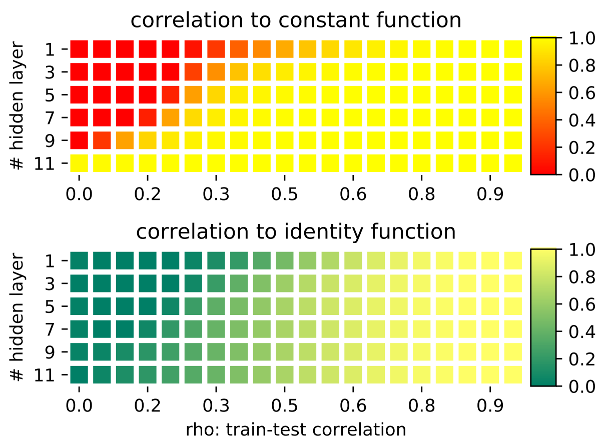


Figure 7: **Visualization of predictions from multi-layer ReLU networks.** The first row shows the test images, and the remaining rows show the predictions from trained multi-layer ReLU FCNs with 1, 3, and 9 hidden layers.



(a) ReLU FCNs



(b) Linear FCNs

Figure 8: **Quantitative evaluation of the learned model on randomly generated test samples at various angles (correlation) to the training image.** The horizontal axis shows the train-test correlation, while the vertical axis indicate the number of hidden layers for the FCNs being evaluated. The heatmap shows the similarity (measured in correlation) between the model prediction and the reference function (the constant function or the identity function). (a) shows the results for FCNs with the ReLU activation function; (b) shows the results for linear FCNs.

E.3 Nonlinear multi-layer fully connected networks

In this section, we consider the general case of multilayer fully connected networks (FCNs) with ReLU activation functions. Figure 7 visualizes predictions from trained ReLU FCNs with 1, 3, or 9 hidden layers. The deepest network encodes the constant map with high confidence; the shallowest network shows behavior similar to that of a linear net. For a quantitative evaluation, we measure the performance via correlation[1] to the two reference functions: the identity function, and the constant function that maps every input to the training point $\hat{x}$. To evaluate the predictions on test images with varying similarity to the training image, we generate the test images according to the following procedure: for a given $\rho \in [0, 1]$, generate a set of test images by sampling random pixels; then add to each image a component parallel to $\hat{x}$ to force the correlation with the training image to be ρ; finally renormalize the test images to match the norm of the training image. So for $\rho = 0$, the test images are orthogonal to $\hat{x}$, while for $\rho = 1$, the test images equal $\hat{x}$. The results are shown in Figure 8. The results for linear FCNs are also shown for comparison. The linear and ReLU FCNs behave similarly when measuring the correlation to the identity function: neither of them performs well for test images that are nearly orthogonal to $\hat{x}$. For the correlation to the constant function, ReLU FCNs overfit sooner than linear FCNs when the depth increases. This is consistent with our previous visual inspections:

[1]See Appendix I for the results in MSE.

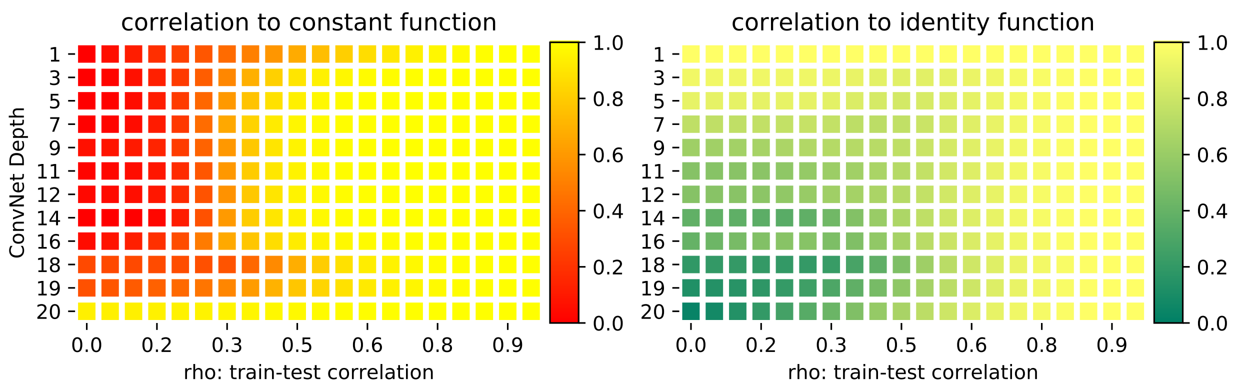


Figure 9: **Evaluation of the predictions of ConvNets on test examples at different angle to the training image.** Heatmap is formatted the same way as in Figure 8.

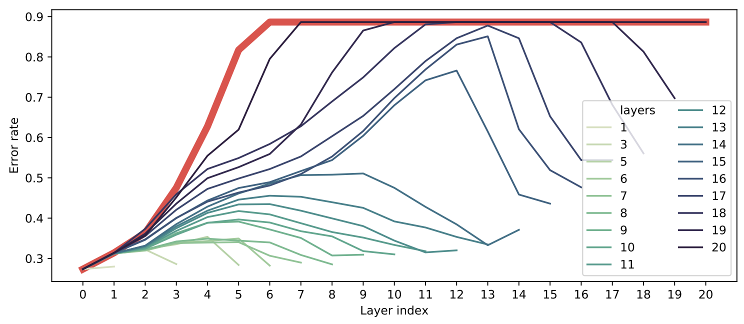


Figure 10: **Measure of the representation collapsing at each layer for trained ConvNets with different depth.** The error rate is measured by feeding the representations computed at each layer to a simple averaging based classifier on the MNIST test set. The error rate at each layer is plotted for a number of trained ConvNets with different depth. The thick semi-transparent red line shows the curve for an *untrained* 20-layer ConvNet for reference.

for shallow models, the networks learn neither the constant nor the identity function, as the predictions on nearly orthogonal examples are random.

F Detailed analysis for ConvNets

Quantitative evaluation for ConvNets with different depth are shown in Figure 9, computed in the same way as Figure 8. The results are consistent with the visualizations: shallow ConvNets are able to learn the identity function from only one training example; very deep ConvNets bias towards the constant function; the ConvNets with intermediate depth correlate well with neither the identity nor the constant function. However, unlike FCNs that produce white-noise like predictions, from the visualization we see that ConvNets behave like edge detectors.

To evaluate how much information is lost in the intermediate layer, we use the following simple criterion to measure the layerwise representations. Take the MNIST dataset, for each layer in the network (trained on a single example), collect the representations obtained by feeding each item in the dataset through the network up to that layer, then do a simple similarity based global averaging classification and measure the error rate. Specifically, the prediction for each test example is the argmax of the mean vector of the (one-hot) training labels, weighted by the correlation between the test example and eeach example from the MNIST training set, computed by the representation from the layer we want to inspect.

This metric does *not* quantify how much information is preserved as the representation propagate through layers in the information theoretical sense, as the information could still be present but encoded in complicated ways that

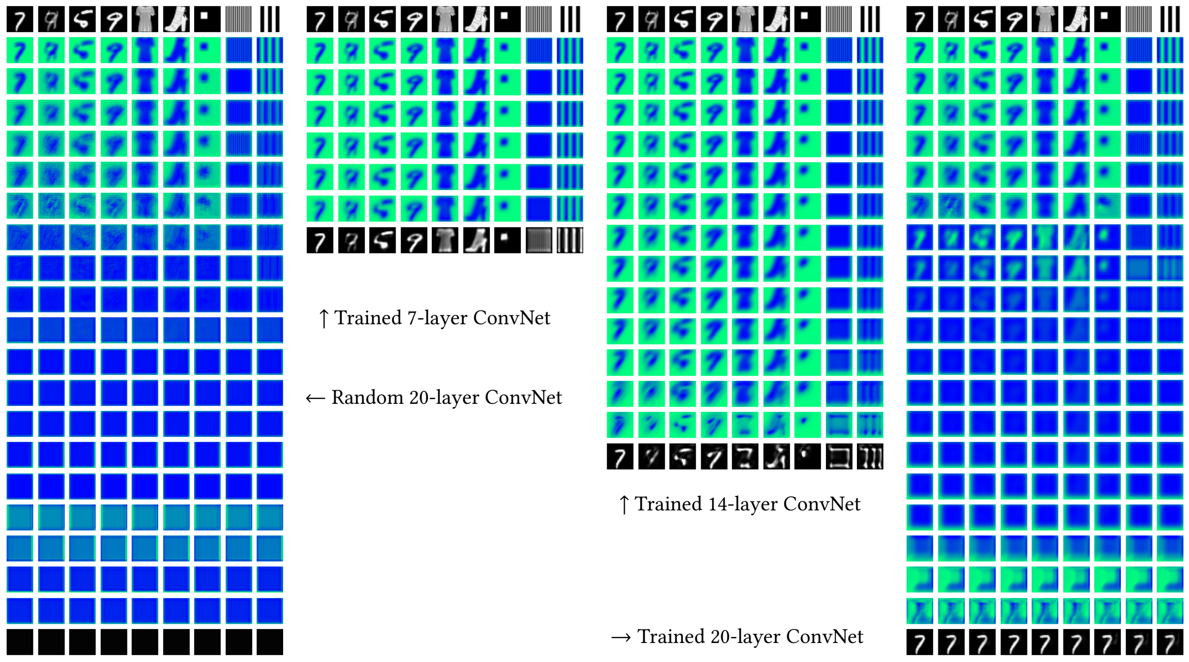

↑ Trained 7-layer ConvNet
← Random 20-layer ConvNet
↑ Trained 14-layer ConvNet
→ Trained 20-layer ConvNet

Figure 11: **Visualization the intermeidate layers of ConvNets with different number of layers.** The first column shows a randomly initialized 20-layer ConvNet (random shallower ConvNets look similar to the truncation of this). The rest of the columns show the trained ConvNets with various number of layers.

makes it hard for simple averaging-based classifier to pick out the signals. But it provides a simple metric for our case: using the raw representation as the baseline, if a layer represents the identity function, then the representation at that layer will have similar error rate to the raw representation; on the other hand, if the a layer collapses to the constant function, then the corresponding representation will have error rate close to random. The results are plotted in Figure 10. The error rate curve for a randomly initialized 20-layer ConvNet is also shown as reference: at random initialization, the smoothing effect makes the representations beyond around layer 5-6 almost useless for our simple classifier. After training, as the error rates for the output layers decrease, the curves generally form "concave" patterns. The trained networks try to recover the smoothed out intermediate layer representations and make connections between the inputs and the outputs. But if the gap is too big to push all the necessary information through, the network will try to infer the input-output relation using partial information, resulting in models that behave like edge detectors. Finally, for the case of 20 layers, the curve shows that the bottom few layers do get small improvements in error rate, but the big gap between inputs and outputs drives the network to learn the constant function instead. It looks like the vanishing gradient problem. However, from Figure 1 we note that given enough training examples, a 20-layer ConvNet can still learn the identity map. More studies on potential vanishing gradient issue can be found in Appendix G.

Because ConvNets preserves spatial structures, we can also qualitatively evaluation the information loss in the intermediate layers by directly visualizing them. We found the behaviors consistent with the observations made here. In particular, in Figure 11, we visualize the intermediate layer representations on some test patterns for ConvNets with different depth. In particular, for each example, the outputs from a convolutional layer in an intermediate layer is a three dimensional tensor of shape (#channel, height, width). To get a compact visualization for multiple channels in each layer, we compute SVD and visualize the top singular vector as a one-channel image.

In the first column, we visualize a 20-layer ConvNet at random initialization[2]. As expected, the randomly initialized convolutional layers gradually smooth out the input images. The shape of the input images are (visually) wiped out after around 8 layers of (random) convolution. On the right of the figure, we show several trained ConvNets with increasing depths. For a 7-layer ConvNet, the holistic structure of inputs are still visible all the way to the top at

[2]Shallower ConvNets at random initialization can be well represented by looking at a (top) subset of the visualization.

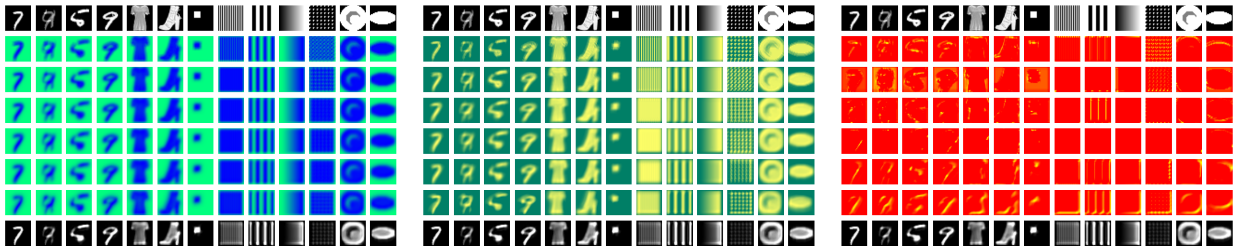


Figure 12: **Visualizing the intermediate layers of a trained 7-layer ConvNet.** The three subfigures show for each layer: 1) the top singular vector across the channels; 2) the channel that maximally correlate with the input image; 2) a random channel, respectively.

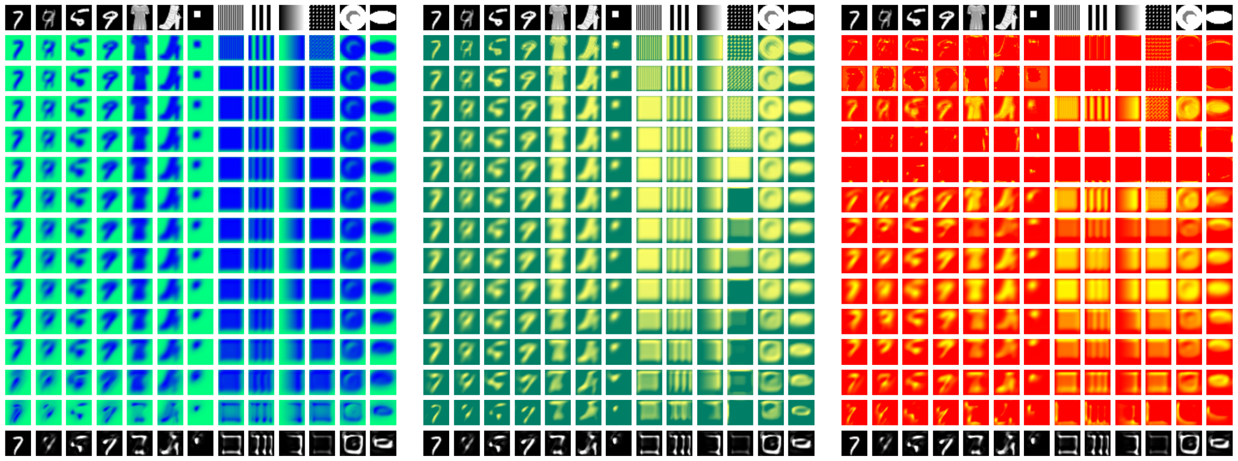


Figure 13: **Visualizing the intermediate layers of a trained 14-layer ConvNet.** The three subfigures show for each layer: 1) the top singular vector across the channels; 2) the channel that maximally correlate with the input image; 2) a random channel, respectively.

random initialization. After training, the network approximately renders an identity function at the output, and the intermediate activations also become less blurry. Next we show a 14-layer ConvNet, which fails to learn the identity function. However, it manages to recover meaningful information in the higher layer activations that were (visually) lost in the random initialization. On the other hand, in the last column, the network is so deep that it fails to make connection from the input to the output. Instead, the network start from scratch and constructs the digit '7' from empty and predict everything as '7'. However, note that around layer-8, we see the activations depict slightly more clear structures than the randomly initialized network. This suggests that some efforts have been made during the learning, as opposed to the case that the bottom layers not being learned due to complete gradient vanishing.

Two alternative visualizations to the intermediate multi-channel representations are provided that show the channel that is maximally correlated with the input image, and a random channel (channel 0). Figure 12, Figure 13 and Figure 14 illustrate a 7-layer ConvNet, a 14-layer ConvNet and a 20-layer ConvNet, respectively.

G Measuring the change in weights of layers post training

In this section, we study the connection between the inductive bias of learning the constant function and the potential gradient vanishing problem. Instead of measuring the norm of gradient during training, we use a simple proxy that directly compute the distance of the weight tensor before and after training. In particular, for each weight tensor W^0

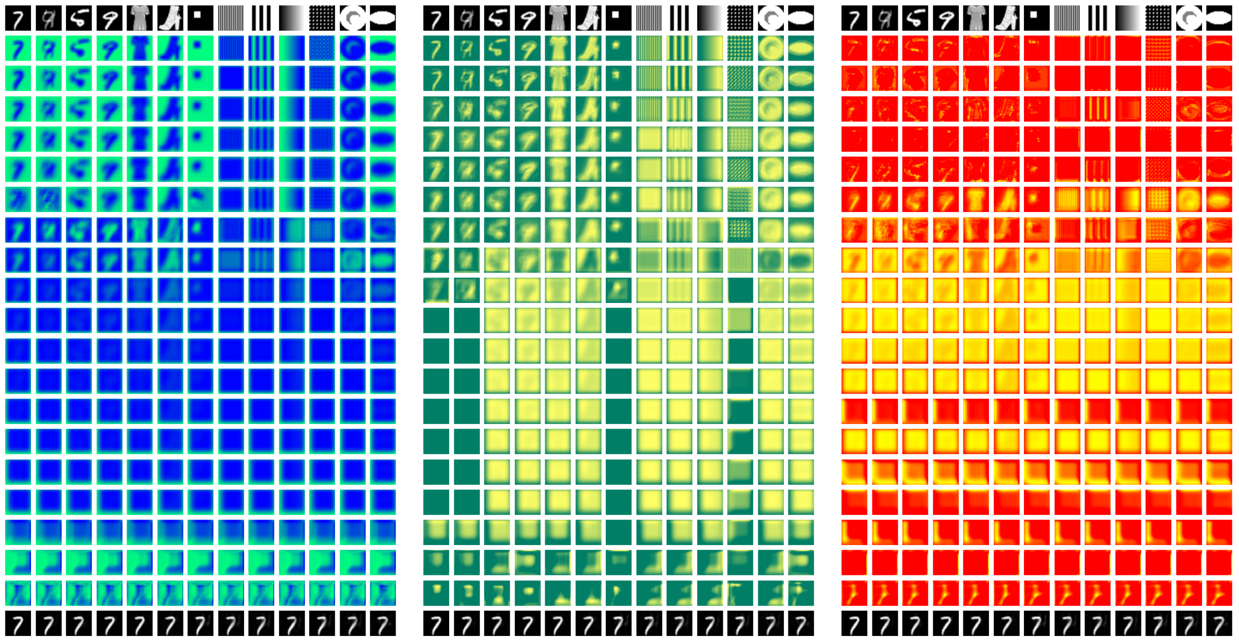


Figure 14: **Visualizing the intermediate layers of a trained 20-layer ConvNet.** The three subfigures show for each layer: 1) the top singular vector across the channels; 2) the channel that maximally correlate with the input image; 2) a random channel, respectively.

and initialization and $W^\star$ after training, we compute the relative ℓ_2 distance as

$$d(W^0, W^\star) := \frac{\|W^0 - W^\star\|_2}{\|W^0\|_2}$$

The results for ConvNets with various depths are plotted in Figure 15. As a general pattern, we do see that as the network architecture gets deeper, the distances at lower layers do become smaller. But they are still non-zero, which is consistent with the visualization in Figure 11 showing that even for the 20-layer ConvNet, where the output layer fits to the constant function, the lower layers does get enough updates to allow them to be visually distinguished from the random initialization.

In Figure 16 and Figure 17, we show the same plots for linear FCNs and FCNs with ReLU activation, respectively. We see that especially for ReLU FCN with 11 hidden layers, the distances for the weight tensors at the lower 5 layers are near zero. However, recall from Figure 7 in Section E.3, the ReLU FCNs start to bias towards the constant function with only three hidden layers, which are by no means suffering from vanishing gradients as the plots here demonstrate.

H Full results on robustness of inductive biases

H.1 Testing on different input image sizes

Due to the perserved spatial structure in ConvNets, we can apply a trained network to a variety of different input sizes. Figure 18 visualizes the predictions of a trained 5-layer ConvNets on images of sizes from 7×7 to 112×112. We found that the learned identity map generally holds up against larger (than the training) input sizes. However, on small inputs, the predictions no longer match well with the inputs. Note ConvNets are in principle capable of encoding the identity function for arbitrary inputs and filter sizes, for example, via the construction in Appendix C.3.

In Section 3 we also test a trained 20-layer ConvNet on various input sizes (Figure 3b). It is interesting that the constant map also holds robustly across a wide range of input sizes. Especially from the prediction on large input images, we found hints on how the ConvNets actually compute the constant function. It seems the artifacts from

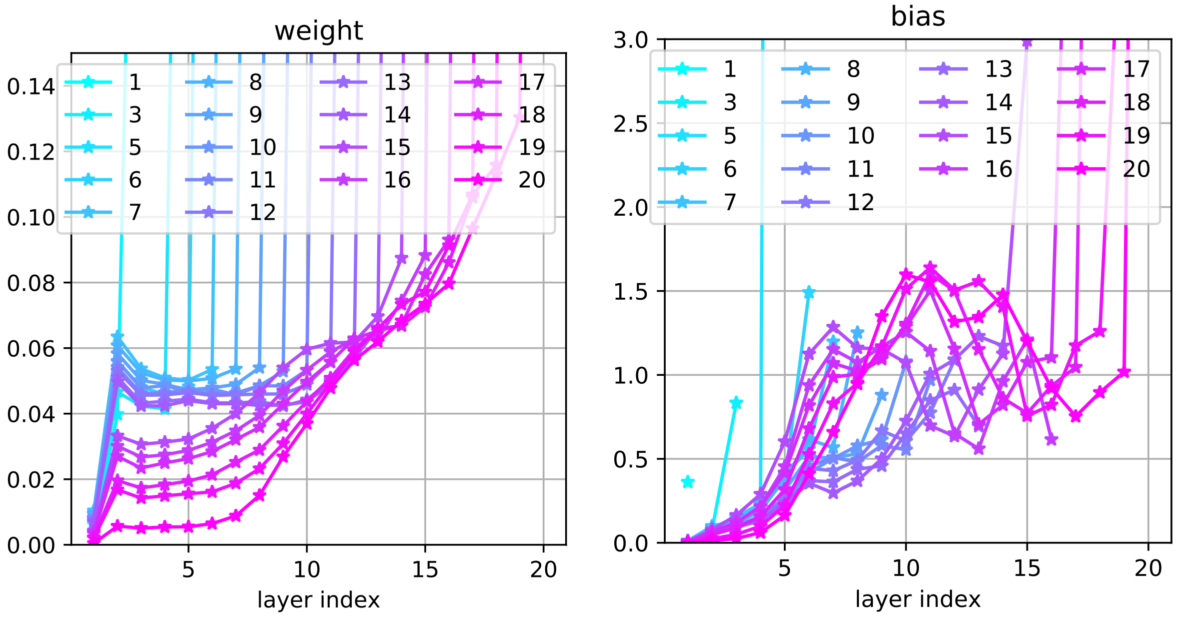


Figure 15: **The relative ℓ_2 distance of the weight and bias tensors before and after training at each layer.** The curves compare ConvNets at different depth. Most of the networks have significantly larger distances on the top-most layer. To see a better resolution at the bottom layers, we cut off the top layer in the figures by manually restricting the y axis.

the outside boundary due to convolution padding are used as cues to “grow” a pattern inwards to generate the digit “7”. The visualizations of the intermediate layer representatiosn in Figure 19 is consistent with our hypothesis: the ConvNet take the last several layers to realize this construction. This is very clever, because in ConvNets, the same filter and bias is applied to all the spatial locations. Without relying on the artifacts on the boundaries, it would be very challenging to get a sense of the spatial location in order to construct an image with a holistic structure (e.g. the digit “7” in our case).

H.2 The upper subnetworks

In the visualization of intermediate layers (Figure 11), the intermediate layers actually represent the “lower” subnetwork from the inputs. Here we investigate the “upper” subnetwork. Thanks again to the spatial structure of ConvNets, we can skip the lower layers and feed the test patterns directly to the intermediate layers and still get interpretable visualizations[3]. Figure 20 shows the results for the top-one layer from ConvNets with various depths. A clear distinction can be found at 15-layer ConvNet, which according to Figure 3a is where the networks start to bias away from edge detector and towards the constant function.

The predictions from the final two layers of each network are visualized in Figure 21. Figure 22 focuses on the 20-layer ConvNet that learns the constant map, and visualize the upper 3 layers, 6 layers and 10 layers, respectively. In particular, the last visualization shows that the 20-layer ConvNet is already starting to construct the digit “7” from nowhere when using only the upper half of the model.

H.3 Varying training factors

The study on the effects of various training hyperparameters on the inductive bias of trained ConvNets is briefly presented in Section 3. The full results are shown in this appendix.

[3]Specifically, the intermediate layers expect inputs with multiple channels, so we repeat the grayscale inputs across channels to match the expected input shape.

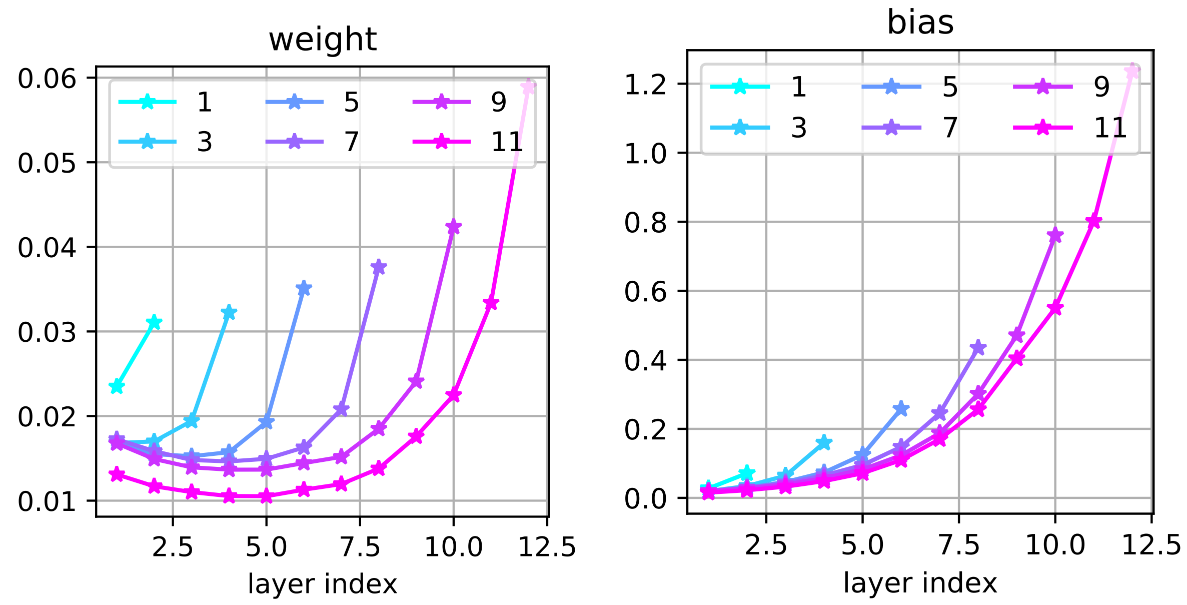


Figure 16: **The relative ℓ_2 distance of the weight and bias tensors before and after training at each layer.** The curves compare linear fully connected networks with different number of hidden layers.

Convolution channel depths Figure 4b in Section 3 shows that the learned identity function is severely corrupted when only 3 channels are used in the 5-layer ConvNets. Figure 23 presents the correlation to the constant and the identity function when different numbers of convolution channels are used. The heatmap is consistent with the visualizations, showing that the 5-layer ConvNet fails to approximate the identity function when only three channels are used in each convolution layer.

Furthermore, Figure 24 visualize the predictions of trained 3-channel ConvNets with various depths. The 3-channel ConvNets beyond 8 layers fail to converge during training. The 5-layer and the 7-layer ConvNets implement functions biased towards edge-detecting or countour-finding. But the 6-layer and the 8-layer ConvNets demonstrate very different biases. The potential reason is that with only a few channels, the random initialization does not have enough randomness to smooth out "unlucky" bad cases. Therefore, the networks have higher chance to converge to various corner cases. Figure 25 and Figure 26 compare the random initialization with the converged network for a 3-channel ConvNet and a 128-channel ConvNet. From the visualizations of the intermediate layers, the 128-channel ConvNet already behave more smoothly than the 3-channel ConvNet at initialization.

Convolution filter sizes Figure 27 and Figure 28 illustrate the inductive bias with varying convolution filter size from 5×5 to 57×57. The visualization shows that the predictions become more and more blurry as the filter sizes grow. The heatmaps, especially the correlation to the identity function, are not as helpful in this case as the correlation metric is not very good at distinguishing images with different levels of blurry. With extremely large filter sizes that cover the whole inputs, the ConvNets start to bias towards the constant function. Note our training inputs are of size 28×28, so 29×29 filter size allows all the neurons to see no less than half of the spatial domain from the previous layer. 57×57 receptive fields centered at any location within the image will be able to see the whole previous layer. On the other hand, the repeated application of *the same* convolution filter through out the spatial domain is still used (with very large boundary paddings in the inputs). So the ConvNets are *not* trivially doing the same computation as FCNs.

Initialization schemes In transfer learning, it is well known that initializing with pre-trained network weights could affect the inductive bias of trained models. On the other hand, different *random* initialization schemes are mainly proposed to help with optimization by maintaining information flow or norms of representations at different depths. It turns out that they also strongly affect the inductive bias of the trained models. Figure 29 visualizes the different

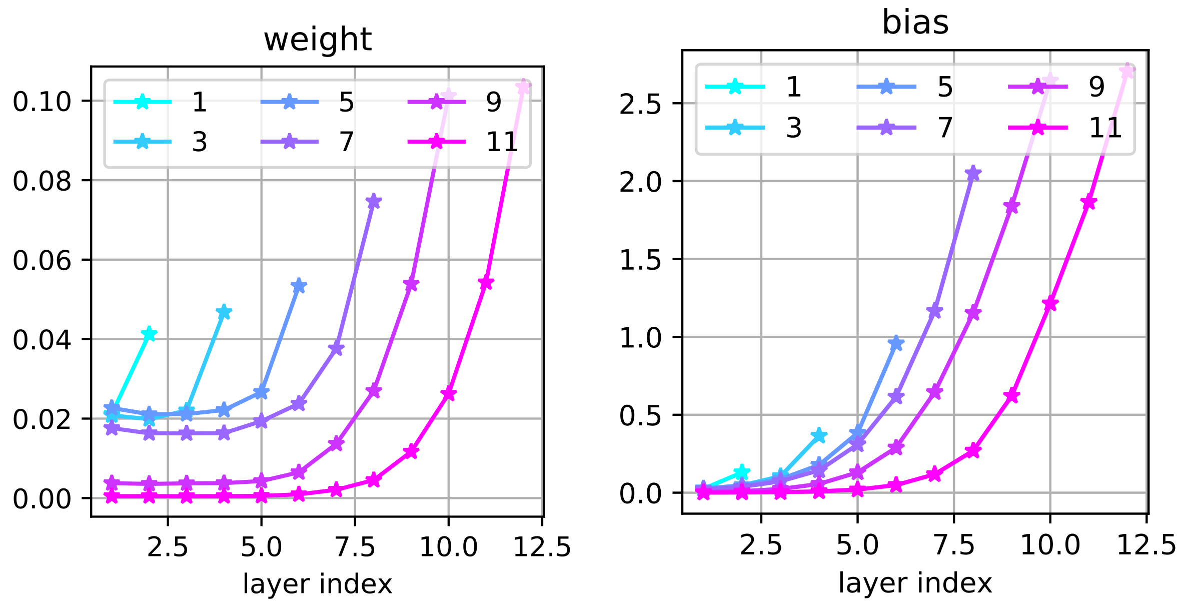


Figure 17: **The relative ℓ_2 distance of the weight and bias tensors before and after training at each layer.** The curves compare fully connected networks with ReLU activation with different number of hidden layers.

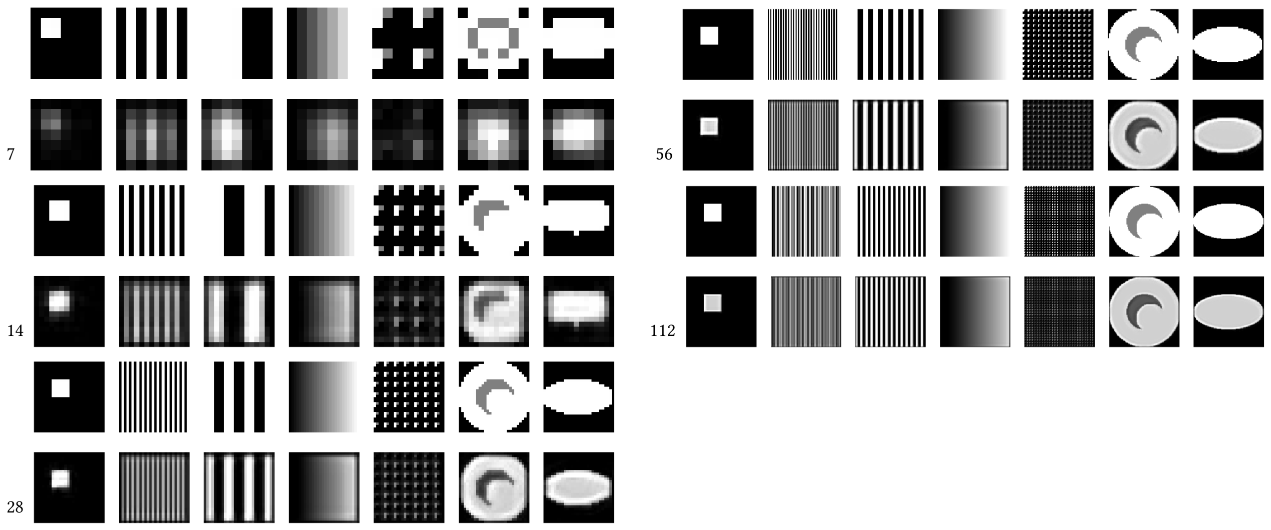
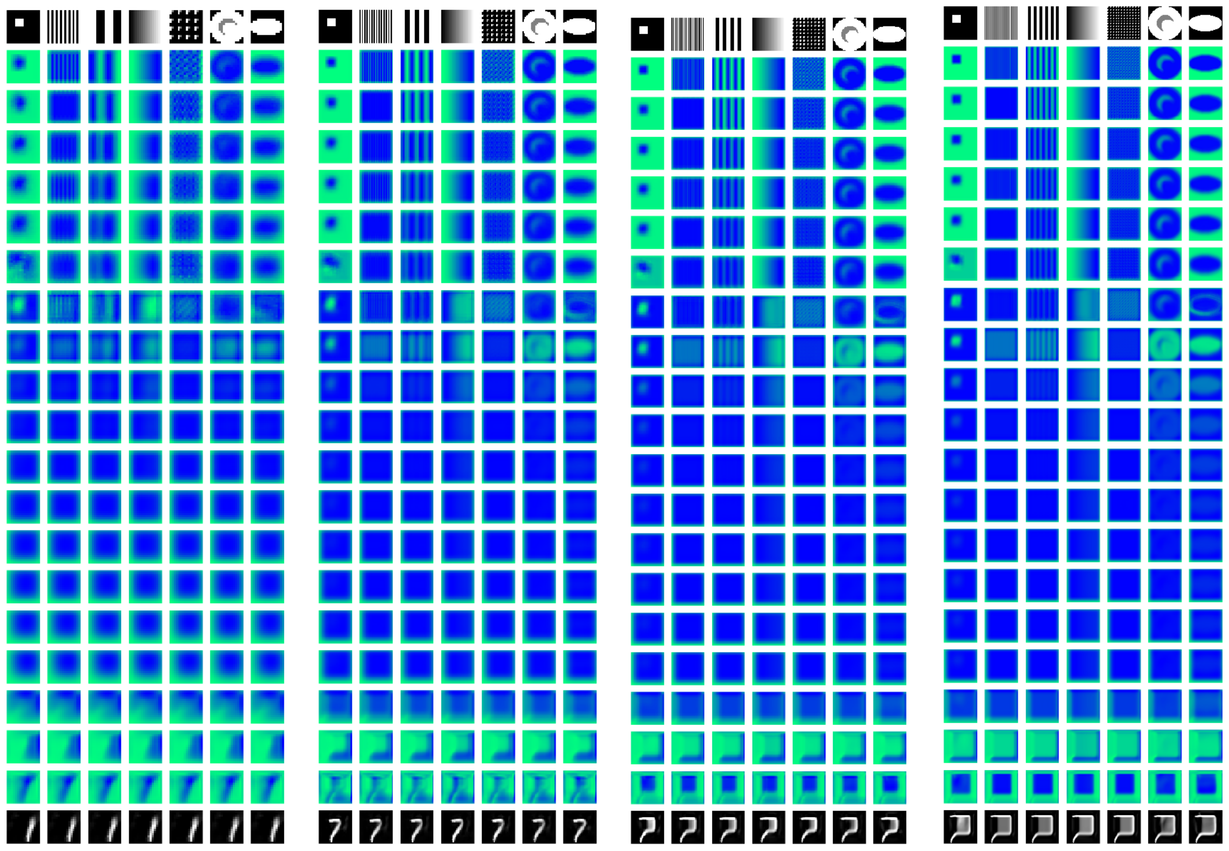


Figure 18: **Visualization of a 5-layer ConvNet on test images of different sizes.** Every two rows show the inputs and model predictions. The numbers on the left indicate the input image size (both width and height).

inductive biases. Let f_i, f_o be the *fan in* and *fan out* of the layer being initialized. We tested the following commonly used initialization schemes: 1) default: $\mathcal{N}(0, \sigma^2 = 1/(f_i f_o))$; 2) Xavier (a.k.a. Glorot) init [13]: $\mathcal{N}(0, \sigma^2 = 2/(f_i + f_o))$; 3) Kaiming init [15]: $\mathcal{N}(0, \sigma^2 = 2/f_i)$. Variations with uniform distributions instead of Gaussian distributions are also evaluated. All initialization schemes bias toward the identity function for shallow networks. But Kaiming init produces heavy artifacts on test predictions. For the bias towards the constant function in deep networks, Xavier init behaves similarly to the default init scheme, though more layers are needed to learn a visually good identity function. On the other hand, the corresponding results from the Kaiming init is less interpretable.



(a) 16 × 16 (b) 28 × 28 (c) 36 × 36 (d) 48 × 48

Figure 19: **Visualization of intermediate representations when testing on images of different sizes from training images for a 20-layer trained ConvNet**. The ConvNet is trained on a 28 × 28 image of the digit "7".

Optimizers First order stochastic optimizers are dorminately used in deep learning due to the huge model sizes and dataset sizes. To improve convergence speed, various adaptive methods are introduced [11, 17, 14]. It is known that those methods lead to worse generalization performances in some applications (e.g. Wu et al. [29]). But they are extremely popular in practice due to the superior convergence speed and easier hyper-parameter tuning than the vanilla SGD. We compare several popular optimizers in our framework, and confirm that different optimizers find different global minimizers, and those minimizers show drastically different inductive biases on test examples, as shown in Figure 30. Some optimizer requires a smaller base learning rate to avoid parameter exploding during training. In particular, we use base learning rate 0.001 for Adagrad and Adamax, 0.0001 for Adam and RMSprop. For comparison, we also include results from SGD with those corresponding base learning rates.

I Correlation vs MSE

Figure 31, Figure 32 and Figure 33 can be compared to their corresponding figures in the main text. The figures here are plotted with the MSE metric between the prediction and the groundtruth, while the figures in the main text uses the correlation metric. Each corresponding pair of plots are overall consistent. But the correlation plots show the patterns more clearly and has a fixed value range of [0, 1] that is easier to interpret.

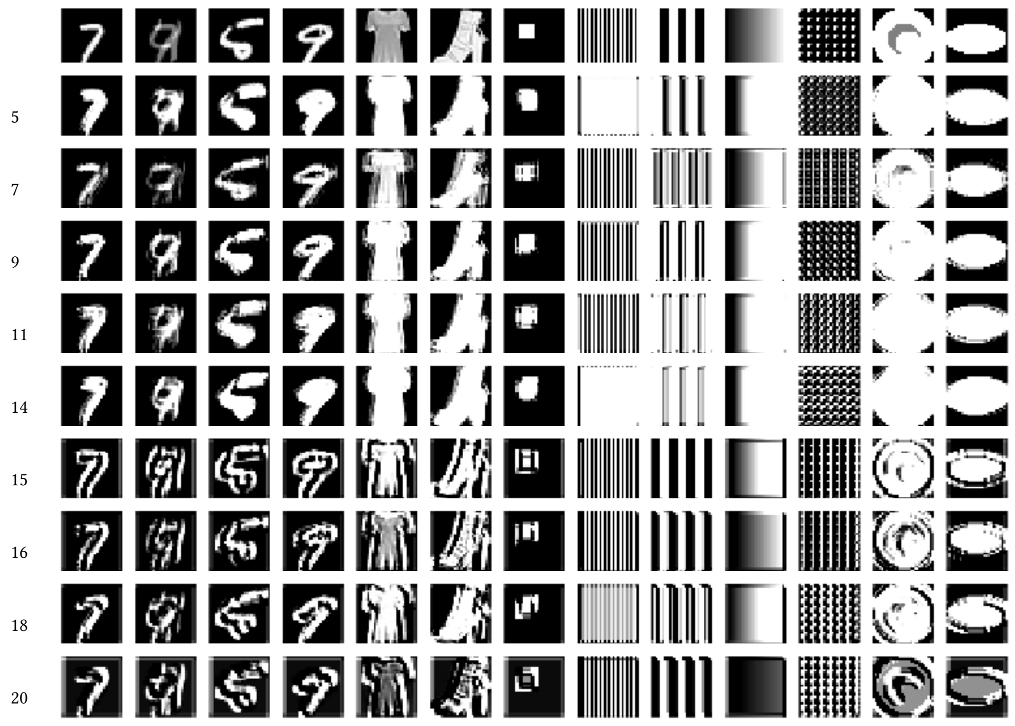


Figure 20: **Visualizing only the final layer in trained networks.** The first row are the input images, which are directly fed into the final layer of trained networks (skipping the bottom layers). The remaining rows shows the predictions from the top layers of ConvNets, with the numbers on the left indicating their (original) depth.

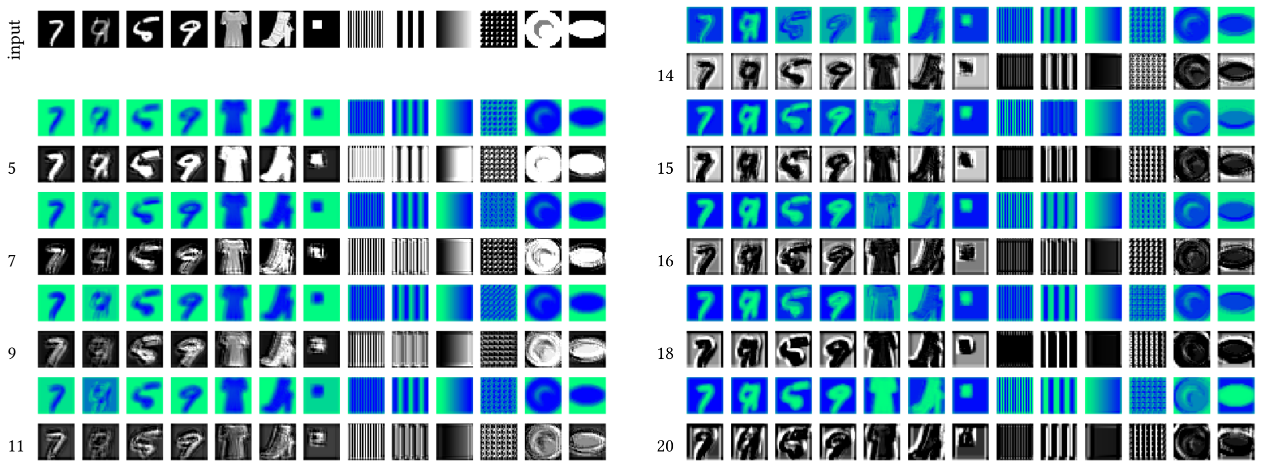


Figure 21: **Visualizing only the top two layers in trained networks.** The first row are the input images, which are directly fed into the top two layer of trained networks (skipping the bottom layers). The remaining rows shows the predictions from the top two layers of ConvNets, with the numbers on the left indicating their (original) depth. More specifically, each of the two top layers occupies one row. The colorful rows are the visualizations (as the top singular vector across channels) of the outputs of the second to the last layer from each network. The grayscale rows are the outputs of the final layer from each network.

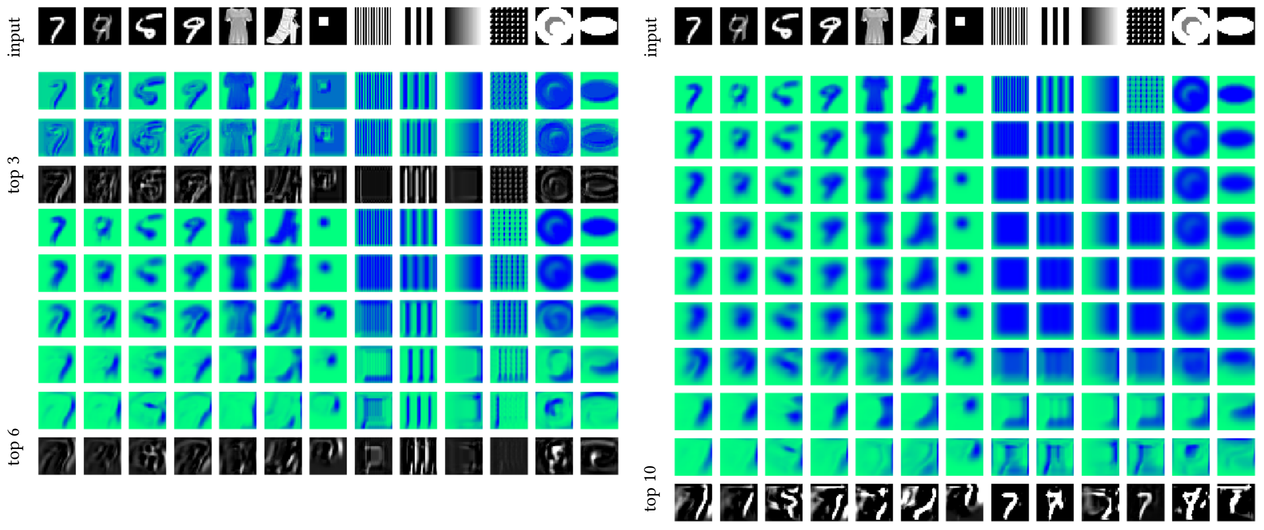


Figure 22: **Visualzing the top 3 layers, 6 layers and 10 layers of a 20-layer ConvNet.** Visualizations formatted in the same way as Figure 21.

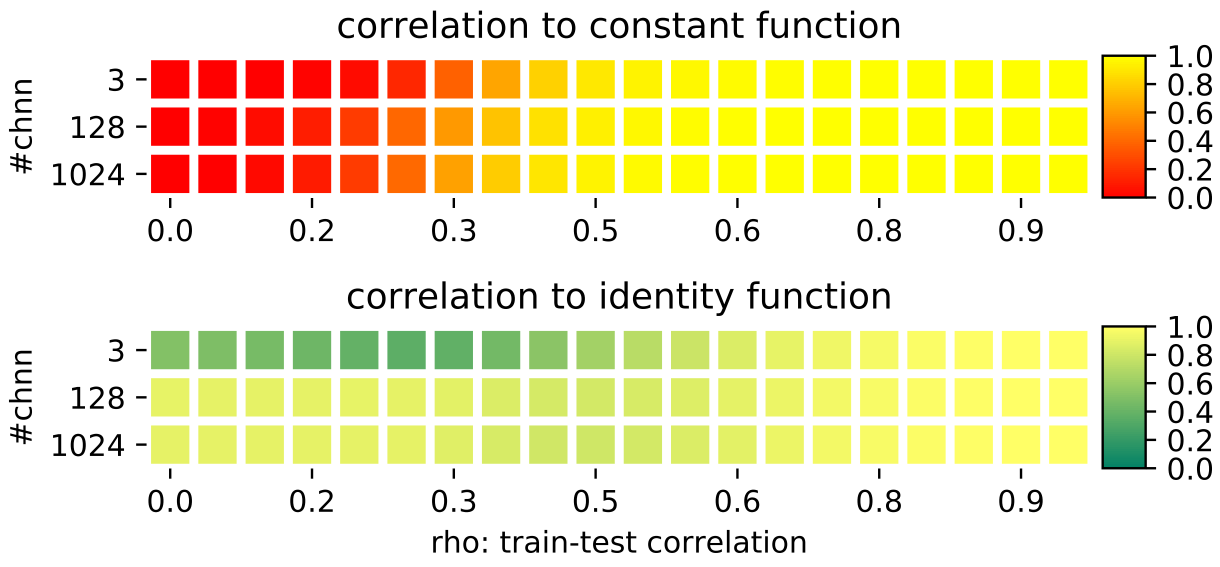


Figure 23: **Correlation to the constant and the identity function for different convolution channels in a 5-layer ConvNet.**

J Learning with multiple examples

We focus on learning from a single example in this paper because the two extreme inductive biases of "the identity map" and "the constant map" can be precisely defined and tested against. When training with multiple examples, the constant map is no longer a viable solution to the optimization problem. Nevertheless, we can still qualitatively evaluate the inductive bias via visualization of predictions. Figure 34 shows the results on various networks trained with two examples. In particular, for networks that are known to overfit to the constant map when trained with one example (e.g. fully connected networks with 9 hidden layers, and 20-layer convolutional networks) demonstrate similar overfitting behavior. In this case, a proper definition of "constant map" no longer holds, as the network perfectly reconstruct each individual digit from the training set. On the test set, a notion of memorization can be

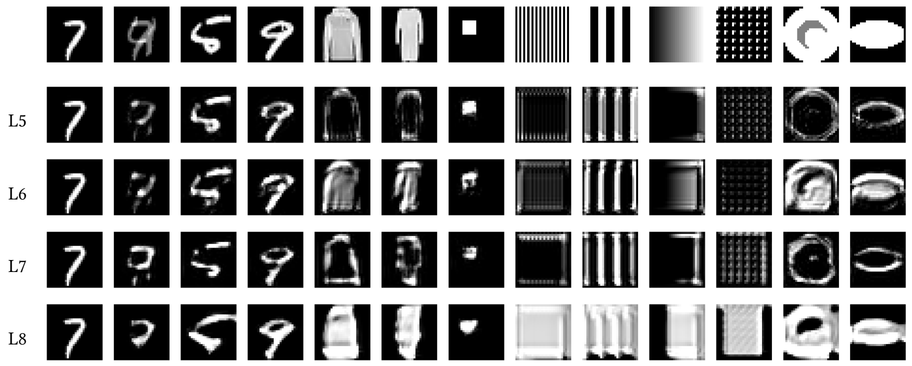
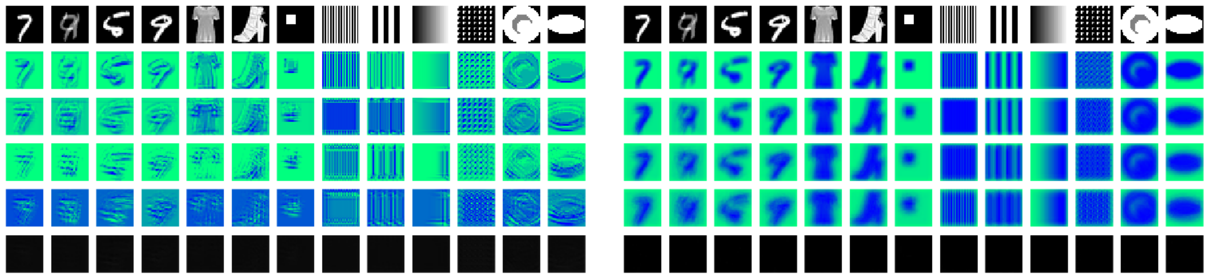


Figure 24: **Visualization predictions from ConvNets with 3 convolution channels and with various number of layers (numbers on the left).** The first row is the inputs, and the remaining rows illustrate the network predictions.



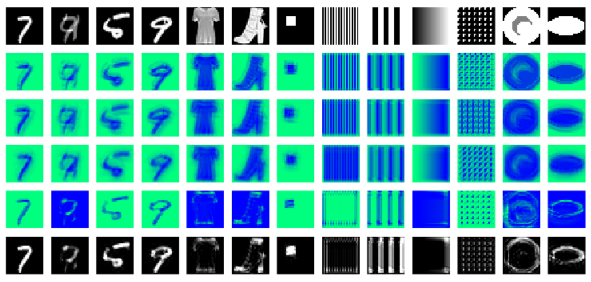
(a) 3 channels, random init

(b) 128 channels, random init

Figure 25: **Visualizing the randomly initialized models to compare two 5-layer ConvNets with 3 convolution channels per layer and 128 convolution channels per layer, respectively.** The subfigures visualize the predictions of intermediate layers of the two network at random initialization. The multi-channel intermediate layers are visualized as the top singular vectors.

recognized as always predicting a mixture of the training samples. Note sometimes the prediction is biased more towards one of the training samples depending on the input patterns.

Figure 35 shows the results with three examples, with similar observations. Figure 36 shows the situation when training with the full MNIST training set (60k examples). In this case, even the deepest convolutional networks we tried successfully learn the identity function. Fully connected networks learn the identity function on the manifold of digit inputs, but still cannot reconstruct meaningful results on other test patterns.


(a) 3 channels, after training

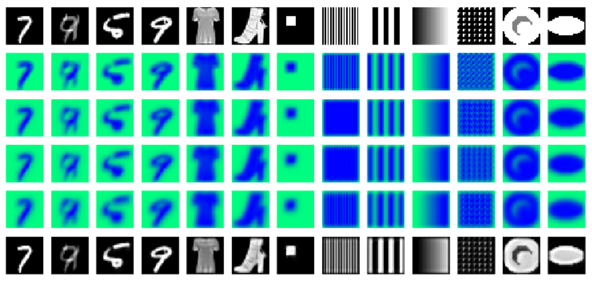

(b) 128 channels, after training

Figure 26: **Comparing two 5-layer ConvNets with 3 convolution channels per layer and 128 convolution channels per layer, respectively.** Layout is similar to Figure 25.

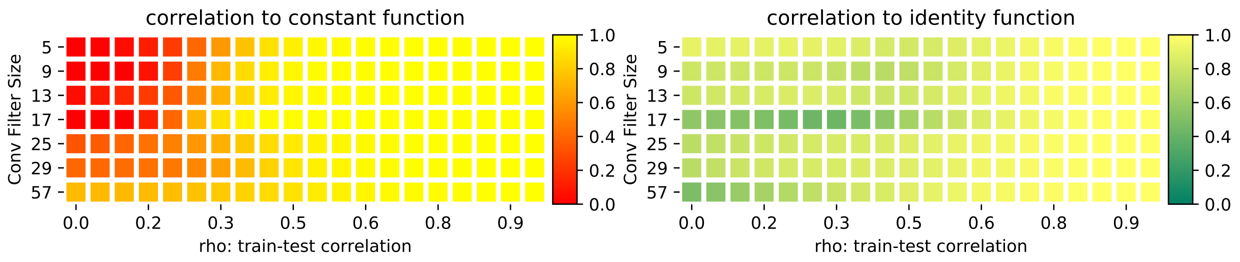


Figure 27: **Inductive bias of a 5-layer ConvNet with varying convolutional filter size.** The heatmap is arranged in the same way as Figure 9, except that the rows correspond to ConvNets filter sizes.

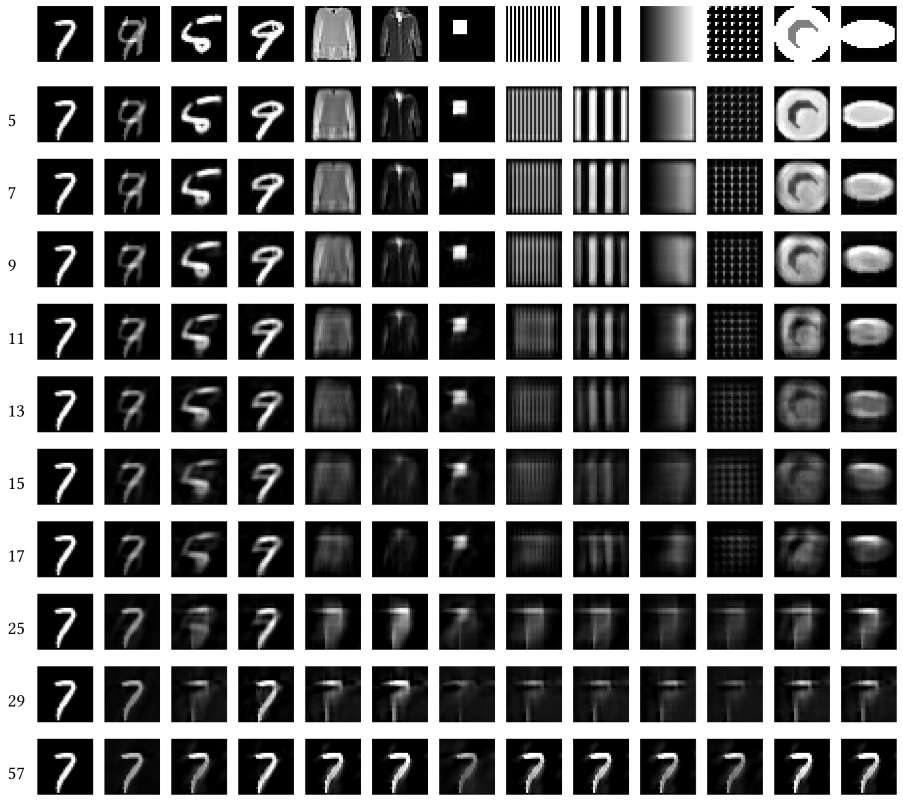


Figure 28: **Visualizing the predictions from ConvNets with various filter sizes.** The first row is the inputs, and the remaining rows are predictions, with the numbers on the left showing the corresponding filter sizes.

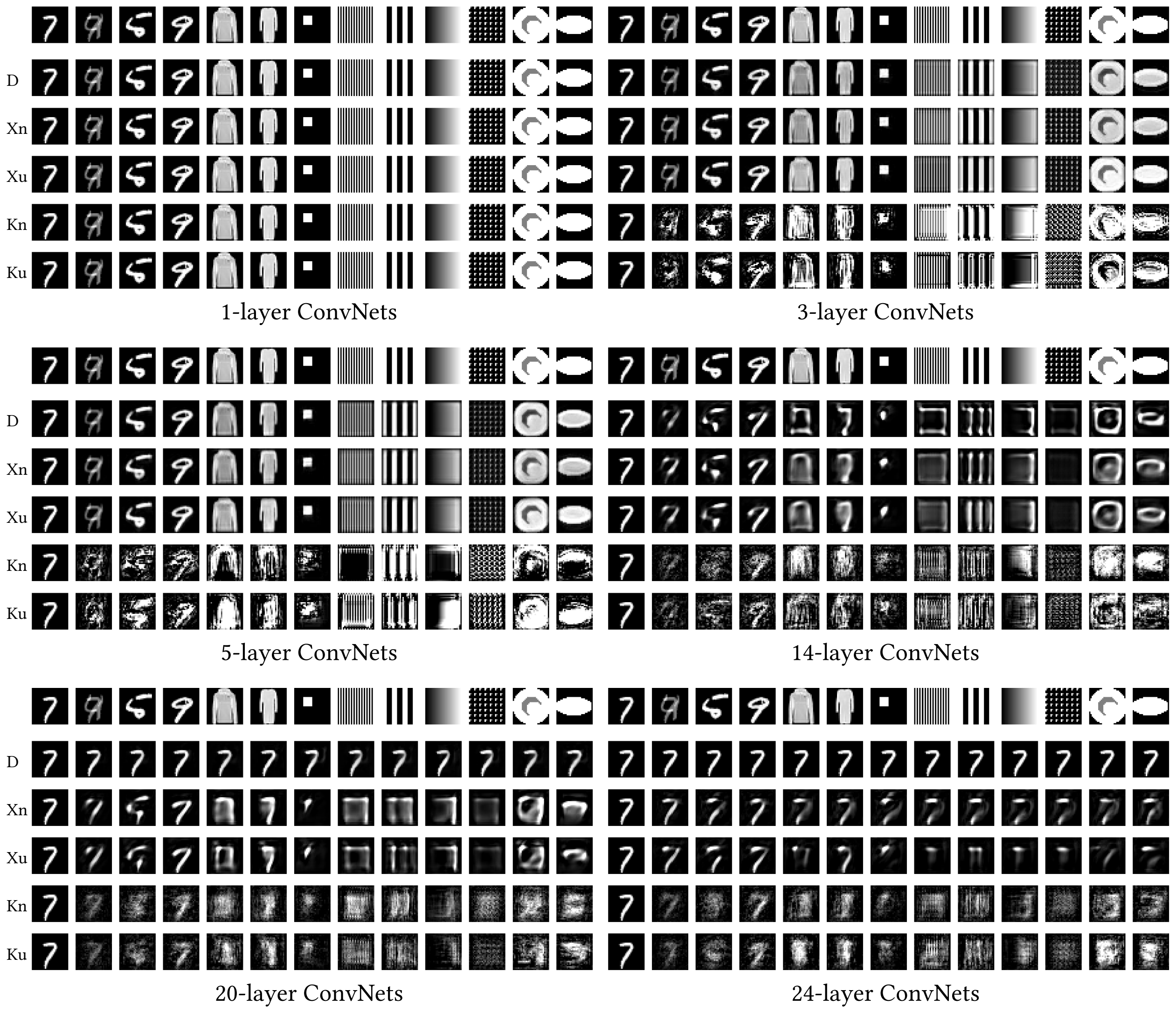


Figure 29: **Visualization of the predictions from ConvNets trained with the D) default, Xn) Xavier normal, Xu) Xavier uniform, Kn) Kaiming normal and Ku) Kaiming uniform initialization schemes.** The first row shows the inputs, and the remaining rows shows the predictions from each trained networks.

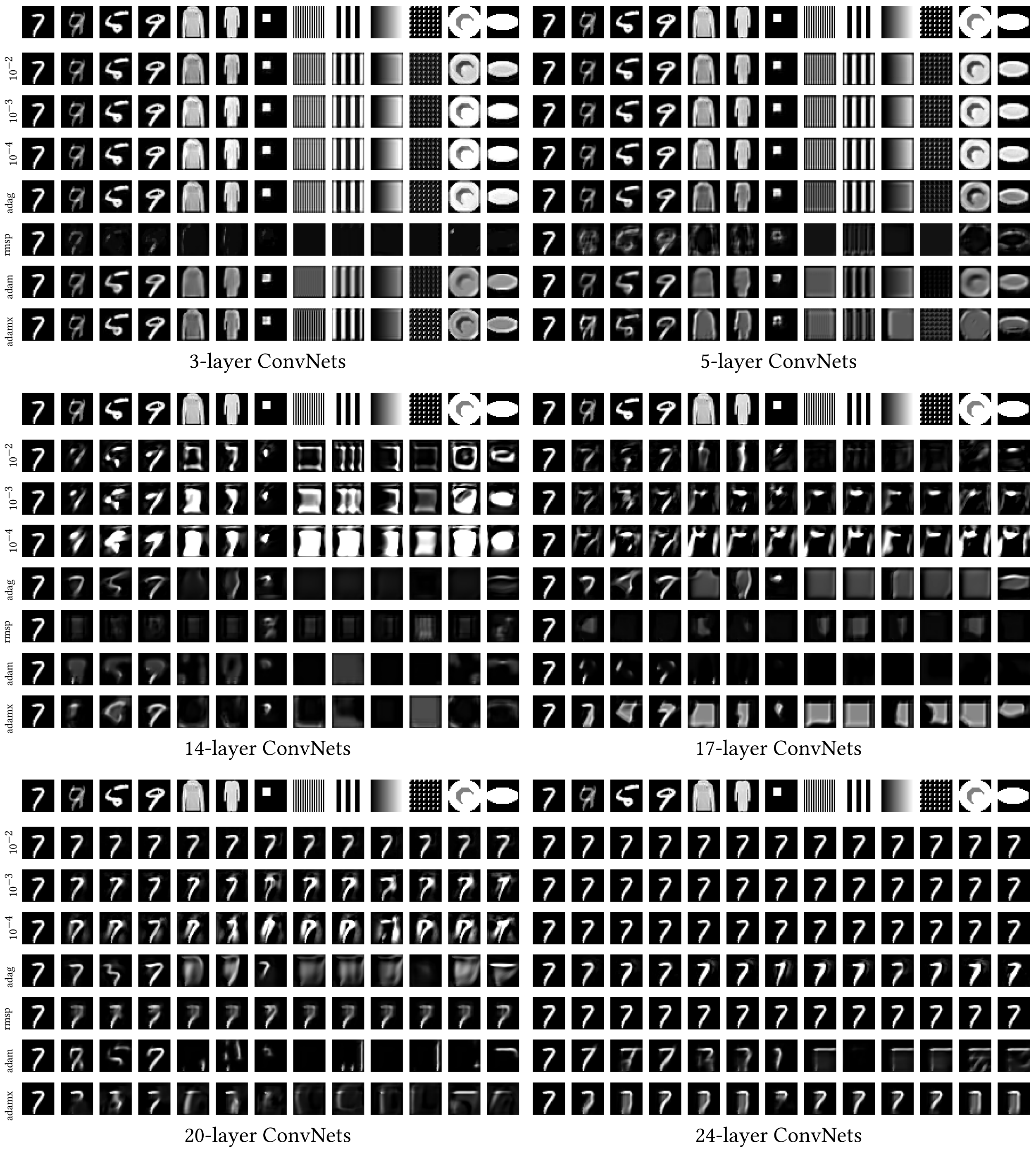


Figure 30: **Visualization of the predictions from ConvNets trained with different optimizers.** The first row shows the inputs, and the remaining rows shows the predictions from SGD (lr 0.01), SGD (lr 0.001), SGD (lr 0.0001), Adagrad, RMSprop, Aam, and Adamax respectively.

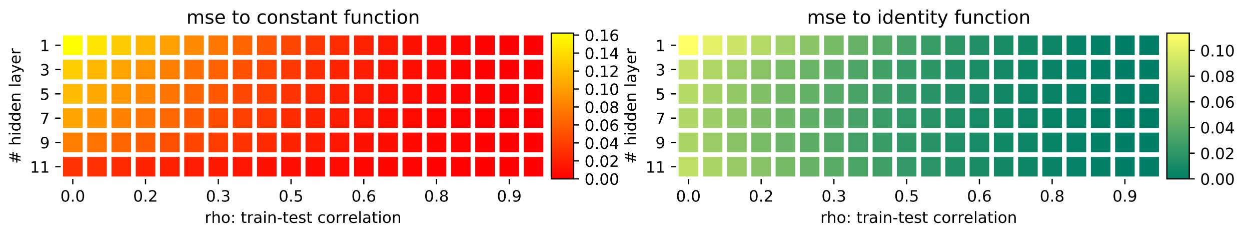


Figure 31: **Quantitative evaluation of linear FCNs.** The same as Figure 8(a), except MSE is plotted here instead of correlation.

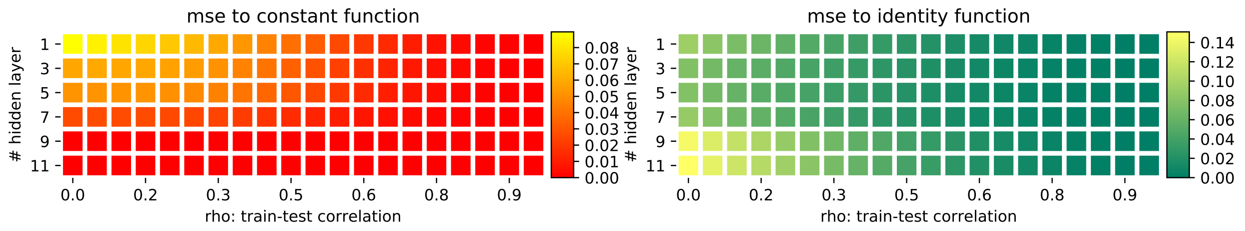


Figure 32: **Quantitative evaluation of ReLU FCNs.** The same as Figure 8(b), except MSE is plotted here instead of correlation.

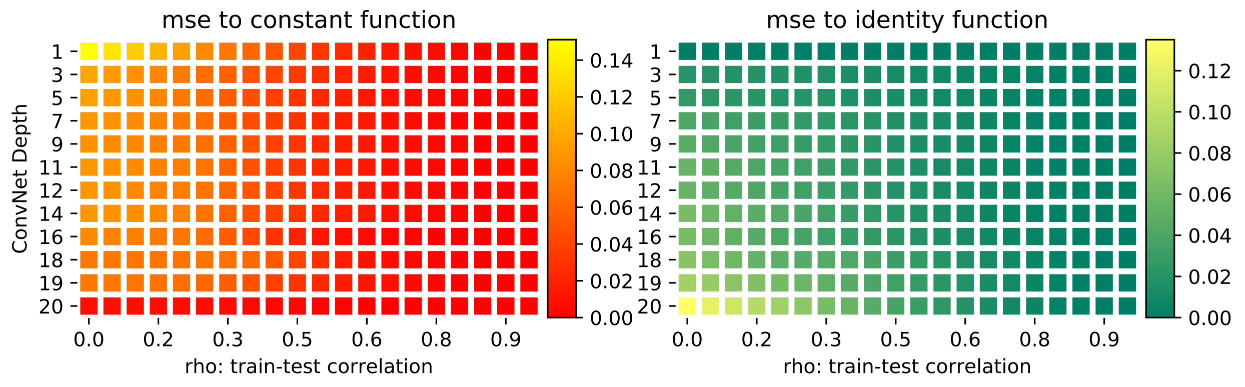


Figure 33: **Quantitative evaluation of ConvNets.** The same as Figure 9, except MSE is plotted here instead of correlation.

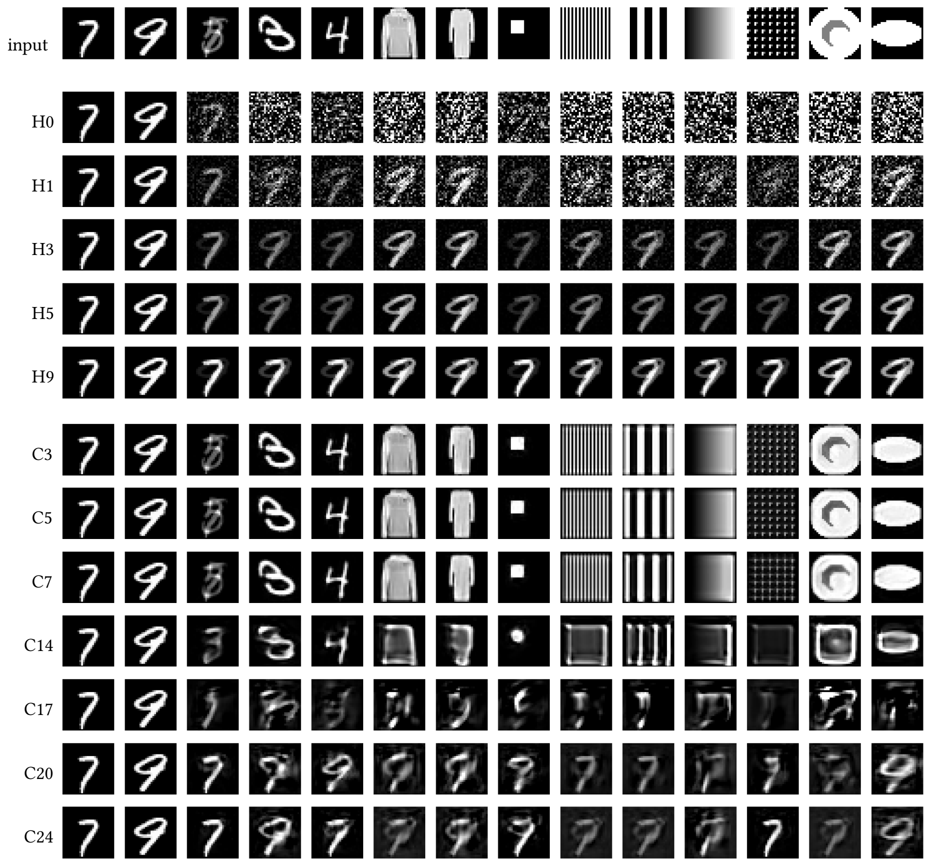


Figure 34: **Visualization of the predictions from networks trained with 2 examples**. The first two columns are training examples, and the remaining columns are unseen test cases. H0—H9 shows fully connected networks with the corresponding number of hidden layers. C3—C24 shows ConvNets with the corresponding number of layers.

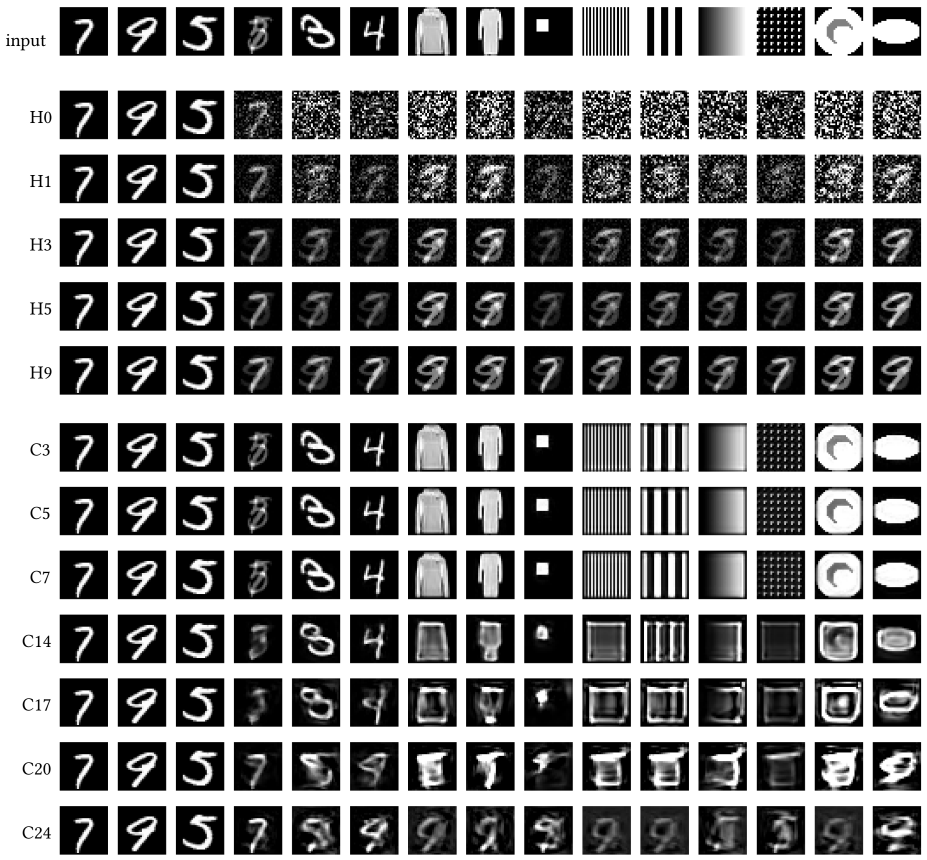


Figure 35: **Visualization of the predictions from networks trained with 3 examples**. The first three columns are training examples, and the remaining columns are unseen test cases. H0—H9 shows fully connected networks with the corresponding number of hidden layers. C3—C24 shows ConvNets with the corresponding number of layers.

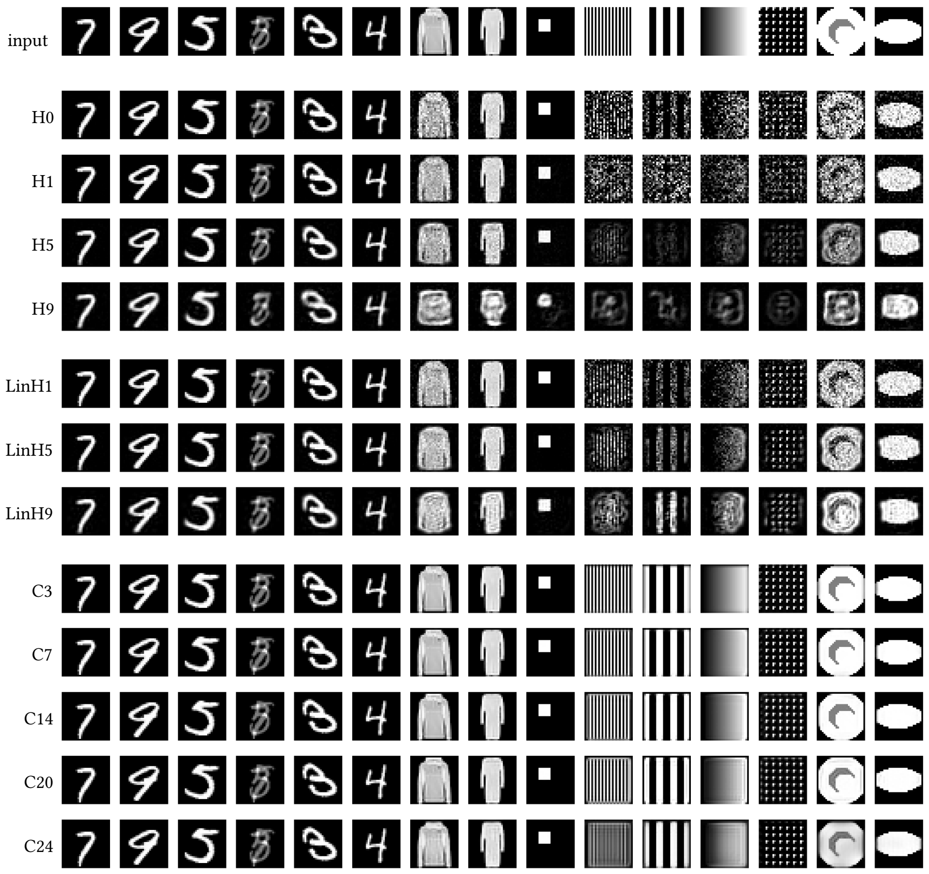


Figure 36: **Visualization of the predictions from networks trained with 60k examples**. The first three columns are (3 out of 60k) training examples, and the remaining columns are unseen test cases. H0—H9 show fully connected networks (ReLU activation) with the corresponding number of hidden layers. LinH1—LinH9 show linear fully connected networks (without activation) with the corresponding number of hidden layers. And C3—C24 show ConvNets with the corresponding number of layers.