Cortically Inspired Machine Learning

NINAI[a–i]

[a]Baylor College of Medicine, Houston, USA; [b]Caltech, Pasadena, USA; [c]Columbia University, New York, USA; [d]Cornell University, Ithaca, USA; [e]Rice University, Houston, USA; [f]University of Toronto, Ontario, Canada; [g]University of Tübingen, Germany; [h]Allen Institute for Brain Science, Seattle, USA; [i]Princeton University, Princeton, USA

Despite major advances in artificial intelligence through deep learning methods, computer algorithms remain vastly inferior to mammalian brains at natural tasks. Here we integrate neuroscience experiments on the mammalian neocortex and computational circuit models based on these data with a theory that the brain embodies a generative model of the world, in order to engineer novel neural networks we call Deep Cortically-Inspired Neural Networks (DeepCINNs). Our new network models perform comparably to conventional artificial networks in standard benchmark tasks, while exhibiting improved performance under challenging natural conditions. Specifically, our DeepCINN model extends conventional networks by incorporating three neurally derived features — unsupervised learning, divisive normalization, and synaptic pruning. These three features produce better performance under the three conditions of limited access to labeled data, online learning, and energetically constrained computation.

Contents

1. Introduction

Improving machine learning through neuroscience.

Artificial Intelligence (AI) has been a dream for a long time, but an elusive one. People have consistently underestimated how hard it is to construct intelligent systems. Famously, in 1955, cognitive scientists proposed to solve AI as a summer project for a few Dartmouth undergraduates. It turned out to be not that easy. Since the 50's there have been many different approaches to solving AI, including expert systems, logic, and symbolic reasoning, evoking alternating excitement and disappointment over the years.

Intriguingly, after decades of exploration, one specific approach to machine learning has emerged as the current winner: deep learning. Multi-layered (deep) artificial neural networks, mimicking real brain circuits, learn how to make accurate inferences from large quantities of training data. These deep networks have notable similarities to the brain, involving many layers, neurons, and plastic synapses that change with experience. This suggests that successful solutions to (artificial) intelligence share essential principles. Yet, deep networks still fall behind mammalian brains in terms of their versatility, learning speed, and robustness. It is possible that this gap is caused by the fact that current deep learning architectures are overly simplistic and resemble real brains only at a very superficial level as they lack of essential elements such as the multitude of cell types, complex neuronal nonlinearities, pervasive feedback, structured connectivity, and local learning rules. This suggests that there are enormous opportunities to revolutionize machine learning and build next-generation AI systems by understanding and incorporating features derived from neuroscience into artificial neural networks.

While the goal of reverse engineering the brain's computational algorithm from neuroscience data is clear, it is not immediately apparent at what level of description this transfer

[1]To whom correspondence should be addressed. E-mail: astolias@bcm.edu, xaq@rice.edu, sinz@bcm.edu

should be attempted. We therefore approach the problem from two points along a spectrum of abstraction, 'bottom-up' and 'top-down'. Our ultimate vision is that these approaches will become two views of one single model: a canonical cortical algorithm used both for predicting neural data and for solving machine learning tasks. We call the machine learning architecture implementing this algorithm Deep Cortically-Inspired Neural Networks (DeepCINNs).

Bottom-up In the bottom-up, data-driven approach we develop a computational module with high neural fidelity that acts as both a good model of cortical computation and as a fully trainable machine learning algorithm. We call this model NetGard (see computational model report). It optimally leverages the unique dataset we are collecting during this project. Detailed anatomical data obtained from dense electron-microscopy reconstructions constrain the architecture of our model, while the complete functional characterization of a cubic millimeter of visual cortex serves both as training data for our computational models and a ground truth for testing its predictions.

The premise underlying this approach is that if the model is able to replicate the brain's evoked activity patterns, then it must be mimicking the brain's representations and transformations, and thus should be able to perform the same computational functions. We therefore ensure that NetGard is both faithful to the biology, and fit for performing hard tasks. Specifically, our model has *structural fidelity* because it uses many elementary building blocks that are functionally matched to neural properties, *functional fidelity* because it can accurately reproduce measured neural activity patterns, and *model fitness* because it is a fully trainable machine learning algorithm that can be applied to new tasks.

We demonstrate in detail in the accompanied computational modeling report that we can already train NetGard to perform machine learning tasks. This is significant because, unlike standard Convolution Networks and many similar models, NetGard is a recurrent neural network with many non-standard neurally-derived properties, for which effective training has never been demonstrated. We show that the neural features offer several advantages without harming performance; in some cases they even improve performance over comparable conventional networks. To summarize briefly, NetGard incorporates distinct excitatory and inhibitory cell types, each with distinctive lateral and feedback connectivity constraints, and basal and apical dendrites compartments with distinct functional properties. With these architectural features, a 2-layer NetGard performs worse than a 4-layer standard ConvNet on object classification in CIFAR10. However, by pre-training it with recorded neural activities, NetGard improves so much on this machine learning benchmark that it outperforms ConvNet. Intriguingly, preliminary results indicate that NetGard may be more robust against fast gradient adversarial attacks than these ConvNets [1].

Top-down It is not clear which cortical circuit elements are responsible for the brain's computational performance. Some may exist only because the algorithm runs on the wetware of the brain. Replicating these elements would unnecessarily increase the number of model parameters, which will be especially problematic as we engineer deeper networks. Thus, ultimately reproducing cortical computation by matching detailed neuronal architecture (i.e. NetGard) may not produce the most efficient algorithm to train and use. We therefore strive to abstract the essential elements of NetGard into another more efficient algorithmic representation, and test them on machine learning benchmarks. To accomplish this goal, in addition to the bottom-up approach we also pursue a top-down, theory-driven approach. Our theoretical framework is built on the premise that the brain is a statistical inference engine that builds and exploits a probabilistic generative model of its environment. Based on this theoretical framework we pursue two complementary directions.

In our first top-down approach, we reverse-engineer generative models that account for a given neural network structure. We call these networks Deep Rendering Models (DRMs). Since these are true generative models, we can use unsupervised and semisupervised learning methods for training, in ways that their discriminative counterparts cannot. Applied to standard convolutional networks, this already yields state-of-the-art results in several visual benchmark tasks. However, the generative models derived from conventional networks remain impoverished compared to the natural world and the brain's internal model of nature. Neuroscience provides us evidence about the brain's internal models, and allows us to build novel generative components that may yield still better performance. Specifically, we reverse-engineer important canonical operations from NetGard and from the published neuroscience literature, and revise our generative models to account for these properties. These generative models form a new generation of DRMs with neural components (DeepCINNs).

During phase I of this project, we demonstrate valuable contributions to machine learning that result from adding three major new neural components into our DRMs: unsupervised learning using feedback, divisive normalization, and neural and synaptic pruning.

1. We incorporate a nonlinear Hebbian-type learning rule to associate feedforward and feedback signals, and use these signals for unsupervised and semisupervised training. Importantly, this allows our networks to achieve state-of-the-art performance when learning from few labeled examples.

2. We include divisive normalization, a prominent property of neural circuits [2] that was automatically learned by NetGard from neural data. The introduction of this neural feature improved performance, learning speed, and learning stability of artificial neural networks on several benchmark tasks particularly in cases where there are a few labeled data (i.e. semi-supervised learning). Moreover, we demonstrate in a variety of network architectures and tasks that divisive normalization can replace the common machine learning trick, called batch normalization, that standardizes the input data across batches of training samples to increase training stability and speed [3]. Because batch normalization processes batches of inputs at once, it cannot be applied in online learning where new data become available one at a time or where the batches are small because of memory limitations. We show that divisive normalization is particularly useful for such online learning scenarios because it can provide a good normalization scheme that enables effective model updates on a trial by trial basis.Accordingly, we believe that our work provides an important contribution towards engineering

deep neural architectures capable of online learning which is an important hallmark for generalized AI. In our DRM framework, divisive normalization naturally disentangles multiplicative scaling of object-specific latent variables, as caused by an illuminant nuisance variable, and the generative framework suggests an avenue for generalizing and possible future improvements.

3. We added neural and synaptic pruning, the process of eliminating neurons or synapses that are not needed to perform target computation. Pruning is pronounced during the development of the neocortex, which is thought to reduce cortical energy demands. Using synaptic pruning, we show benefits in our artificial networks, compressing the neural network by 60% with minimal change in performance. This controls model complexity, saves computation time and energy, both at a premium during hard tasks.

NetGard exhibits many other interesting features. Thus, we plan to reverse-engineer NetGard and incorporate more of its features into our DRM machine learning model, including hierarchical divisive normalization, cell-type-specific connectivity patterns, multiplicative feedback gating in apical dendrites, recurrent dynamics and lateral connections, and use of feedback during inference. Subsequent work during Phase II will also make connections between plasticity rules and NetGard and DRM models, including new cell-type specific rules we measured experimentally.

In our second top-down approach, we are developing methods to infer nonlinear dynamics of neural activity and interpret these dynamics as the operation of a message-passing algorithm. Message-passing is a class of algorithms that performs a global computation by applying local operations on a graph. In this application, the computation is probabilistic inference, and the graph is the interaction graph of a probabilistic graphical model. The resultant nonlinear functions can be implemented in a novel hybrid neural network that adopts the large-scale architecture from existing deep generative models or deep neural networks, but modifies the nonlinear operations according to novel message-passing rules. We hypothesize that this computational structure operates not on the level of individual neurons, but on the level of neuronal populations — on the information that the distributed activity patterns encode. We have developed tools for inferring inference algorithms from neural population data evoked by visual stimuli. We are now applying these tools to responses to dynamic naturalistic visual movies that we have designed. The movies are optimized to reveal canonical message-passing updates expected from lateral and proximal feedback interactions, since these are the types of interactions that we can directly measure with our experiments. We have begun to apply these tools to identify message-passing algorithms within population activity for both recorded neural data and NetGard. Since the inferred message-passing algorithm runs on a probabilistic graphical model, we may benefit from incorporating the probabilistic graphical model structure of DRMs obtained by reverse-engineering NetGard. Finally, to model the process of learning within the message-passing framework, we will track the slow low-level changes in a next-generation NetGard undergoing synaptic plasticity, and infer the corresponding high-level changes at the graphical model.

Our combined bottom-up and top-down approaches allow us to synthesize multiple forms of evidence from experimental neuroscience, computational neuroscience, and machine learning, and to select the best level of abstraction to transfer knowledge from complex neuroscience experiments to revolutionary machine learning algorithms.

2. Linking Generative Models and Deep Neural Networks

Theoretical framework.

Our brains evolved to interpret sensory observations and infer their causes – properties of an unobservable external world. Acting in and upon this world requires perception, motor control, and learning. In visual perception, for example, we have to infer the causes of the light patterns we observe, such as the presence of a friend in the distance or the identity of an object. A major challenge is that these observed data are inherently ambiguous: many objects are compatible with any given retinal image, while conversely any one object can produce many diverse images, owing to such processes as illumination, occlusion, and projection from 3D to 2D.

Mathematically speaking, probabilistic (Bayesian) inference is the right way to solve these problems [4, 5]. Inference here means integrating all evidence to find information about variables of interest. The optimal solution requires a model that describes statistical relationships between observations and unobservable causes (hidden or latent variables). This statistical model is known as a generative model, and efficiently summarizes how the observations are generated, or caused, by latent variables. These latent variables are crucial because successful performance in interesting tasks generally requires inferring them and choosing actions based on them.

Unfortunately, while the mathematical concepts are clear, optimal inference is impossible in practice, as it requires exponentially large resources in storage, computation time, and energy. Computational constraints thus make optimal inference prohibitive. Practical inference requires an agent to learn and use approximate models, approximate inference, or both, to accomplish a given task.

A key ingredient to this simplification is that not all configurations of variables are equally likely – or even possible. There is rich structure in the world, and typically each latent variable directly affects only few others. In our theoretical framework, this core assumption manifests as a probabilistic graphical model [6] – a description of a joint probability distribution where all variables are depicted as nodes on a hypergraph and hyperedges represent conditional interactions between them. For example, a typical visual scene is composed of multiple objects composed of interacting parts and subparts, each having properties like pose and texture. Latent variables represent the objects, parts, and their properties; conditional probability distributions reflect how they interact.

Behavioral evidence suggests that the brain performs general purpose inference more successfully than any other computer, and we aim to find out how. Our team is learning, understanding, and applying the approximation tricks embodied by neurobiology.

We have taken an integrated, multifaceted approach to using experimental data to create new algorithms, by combining functional circuit modeling with machine learning methods, novel statistical analyses, and principled theories of inference

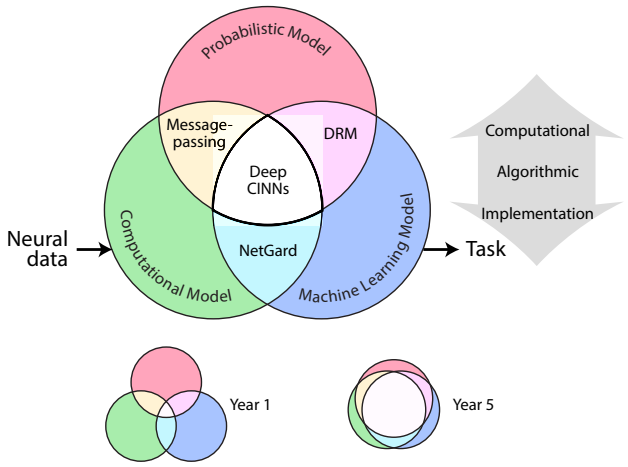


Fig. 1. Summary schematic of our efforts to transfer knowledge from neuroscience data to instantiated machine learning algorithms on benchmark tasks. Our approaches merge probabilistic models, computational models, and machine learning models to synthesize a new Deep Cortically-Inspired Neural Network (DeepCINN) that solves inference tasks in novel ways. Our approaches emphasize the three types of models in different ways, varying in the level of explanation [7] from the computational level to the algorithmic level to the implementation level, with varying degrees of abstraction. NetGard is a bottom-up computational neural model constrained by neural data, that can also be trained directly as a machine learning model. Message-passing is a top-down probabilistic model that abstracts nonlinear operations from the computational model responses or neural data. The Deep Rendering Model is a probabilistic model with neurally inspired features that compiles into a trainable machine learning model. Through concerted interactions of our teams over time, features of these different models will be integrated to converge toward a single synthesized machine learning algorithm, the DeepCINN. This resultant network provides the main inference engine for the benchmark tasks.

in probabilistic generative models (Fig. 1). Next we briefly outline these components and how they interact in our framework, enabling us to create novel Deep Cortically Inspired Neural Networks (DeepCINNs; Fig. 1).

Our approach to generative models begins with the observation that successful modern deep neural networks implement interpretable approximate inferences on an approximate generative model of images [8]. We call this underlying model family the Deep Rendering Model (DRM). In our approach, we identify a generative model for which an inference algorithm (exact or approximate) reproduces the specific architectural and nonlinear operational constraints of a particular deep neural network. As we discuss below, we have successfully applied this method to reverse-engineer generative models for a variety of popular deep neural network architectures such as Deep Convolutional Networks (DCNs) and random forests [8].

However, the brain's cortical circuits are not merely deep convolutional networks and deriving the brain's generative model is an important step towards solving some some of the key problems that current artificial neural networks still have. For instance, they display catastrophic failures of inference upon presentation of 'adversarial' examples [9] and they generalize poorly in situations where only few training examples are available. The brain, however, is largely immune to these problems. Presumably, these enviable computational properties originate from a more accurate model of the world and a more reliable inference algorithm than any artificial agent yet built.

Deep Rendering Mixture Model framework.

A fundamental hypothesis of our work is that deep neural networks (DNNs) in the brain are performing probabilistic inference with respect to a generative probabilistic model of the world [10]. But how can we link such DNNs to generative models? Here we describe a framework for establishing the link via a *reverse-engineering* approach, that has been successful with several classes of *artificial* DNNs, including deep Convnets (DCN), Autoencoders, and LSTM/GRU Recurrent Neural Networks (RNNs) (work in progress). Given a specific artificial DNN architecture, the approach finds a generative model such that a particular type of inference in that model is equivalent to the DNN of interest. Note that this need not always be possible; not all DNNs (artificial or real) have underlying generative models that are natural or simple to describe. And yet our initial successes with state-of-the-art DNN architectures suggests it is possible for the most popular and successful DNN architectures to date, including DCNs.

Deep Convnets have driven tremendous advances in machine vision tasks, sparking a revolution in both academia and industry. And yet, despite their success, how they work – and why they fail when they do – remain poorly understood. Recent work from our labs [11–13] have shown that the above approach can be employed successfully on DCNs, resulting in a hierarchical generative model that we call the *Deep Rendering Mixture Model (DRMM)*.

Compared to a discriminative classifier model, there are many benefits to having a generative classifier model.

- We can perform supervised, unsupervised, and semi-supervised learning in a unified framework. Current DNNs are trained on enormous sets of labeled examples. Labeling examples, however, is time consuming and potentially costly.

- We can perform top-down inference and learning in a principled way, using the Expectation-Maximization (EM) algorithm. Typically, such learning is as fast or faster than discriminative counterparts, due to the stronger distributional assumptions imposed by the generative model.

- We can use principled metrics (such as model log-likelihood) to assess the fitness of a model and to perform model selection. In light of our generative modeling approach to neural nets, this enables us to learn how many neurons and layers are needed in order to achieve high performance and generalization on the task.

- We can extend the static generative model to a dynamic one in a principled way, by allowing for probabilistic transitions in the latent variables (e.g. Hidden Markov Model).

- We can sample from the model to see how well it captures key aspects of the inputs.

However, there are a few limitations. In particular, generative models generally impose more stringent distributional assumptions which can lead to worse fits compared to discriminative counterparts when the model mismatch is strong [14]. Discriminative models are consistent with many generative counterparts, and as such are more flexible and robust to model

mismatch, a key reason for their outstanding performance in practice [15, 16].

In the end, the many benefits of generative models seem to outweigh the few disadvantages. As we will see, they are essential to a good performance and generalization in a semi-supervised learning setting..

The Deep Rendering Mixture Model (DRMM): a graphical model for Deep Convnets.

The DRMM is a hierarchical generative model in which the image is generated iteratively in a coarse-to-fine manner. It has been shown that the bottom-up inference in a variant of the DRMM (after a discriminative relaxation), corresponds to the feedforward propagation in the DCNs. The DRMM enables us to perform semi-supervised learning with DCNs. Training results for semi-supervised learning with the DRMM are discussed below.

The *Deep Rendering Mixture Model (DRMM)* is a deep Gaussian Mixture Model (GMM) with specialized structure in the latent variables and the parameters. Two key features of the DRMM are that (i) it explicitly captures variation due to nuisance transformations via latent variables and (ii) the different components of the GMM share parameters. For example, in an object recognition task, such nuisance variation might be due to a change in pose of an object.

Shallow Rendering Model For clarity, we will start with a shallow version of the DRMM, the *Shallow Rendering Mixture Model (SRMM)*. The generation of an image sample according to the SRMM takes the form:

$$\begin{aligned} c \sim \text{Cat}(\{\pi_c\}), \quad g \sim \text{Cat}(\{\pi_g\}) \\ \mu_{cg} \equiv \Lambda_g \mu_c \\ I = \mu_{cg} + \eta, \\ \eta \sim \mathcal{N}(0, \sigma^2 \mathbf{1}_{D^{(0)}}) \end{aligned}$$

where $c \in \mathcal{C}$ is the object category, $g \in \mathcal{G}$ are the latent (nuisance) variables, π_c, π_g are mixing probabilities and Λ_g are parameters that encompass the nuisance transformations. Here the image $I \in \mathbb{R}^D$ is generated by adding isotropic Gaussian noise η to a "rendered" template μ_{cg}. μ_c is a learned template for category c.

The latent variables g determine the nuisance transformation modeled here as a linear transformation Λ_g, and composed of a collection of latent variables g_x, one per location or region $x \in \mathcal{X}$ in the image. Each individual g_x is composed of two latent variables, a binary masking variable $a_x \in \{0, 1\}$ that controls whether or not to render at location x, and a translational nuisance variable t_x which specifies the fine-scale position of the rendered template (e.g. within a 2x2 patch). More details about the definition of the latent variables g and their action via Λ_g are available in [12].

Inference task: Find the most probable configuration. In classification, the task is to predict the category c from a given image I. In the generative models, this is called *inference* because we infer variables of the model from the image. The most common method of inference is to compute the most probable configuration (MPC) of the latent variables c, g given the image (MPC is also called the joint Maximum *a posteriori* or joint MAP). In [12], it is shown that inference of the MPC yields the ReLu nonlinearity (from $\max_{a_x}$) and spatial max-pooling (from $\max_{t_x}$). Thus, we have achieved our goal of reverse-engineering, by reproducing the structure of a single layer in a Convnet as inference in a generative model.

Going Deep From the SRMM we can build a deep model (DRMM) by stacking single SRMM layers into a hierarchy. The key changes are that the rendered template μ_{cg} is now the result of the composition of many nuisance transformations further up in the hierarchy. This allows the model to capture nuisance variation at different length scales as each layer ℓ in the hierarchy represents one length scale (or equivalently, receptive field size). Importantly, the nuisance transformations are now conditioned on latent variables at each length scale $g^{(\ell)}$. All together, generation of an image according to the DRMM takes the form:

$$\begin{aligned} c^{(L)} \sim \text{Cat}(\{\pi_{c^{(L)}}\}), \quad g^{(\ell)} \sim \text{Cat}(\{\pi_{g^{(\ell)}}\}) \; \forall \ell \\ \mu_{cg} \equiv \Lambda_g \mu_{c^{(L)}} \equiv \Lambda^{(1)}_{g^{(1)}} \Lambda^{(2)}_{g^{(2)}} \dots \Lambda^{(L-1)}_{g^{(L-1)}} \Lambda^{(L)}_{g^{(L)}} \mu_{c^{(L)}} \\ I = \mu_{cg} + \eta, \\ \eta \sim \mathcal{N}(0, \sigma^2 \mathbf{1}_{D^{(0)}}) \end{aligned}$$

where $\ell \in \{1, 2, \dots, L\}$ is the layer, $c^{(L)}$ indexes the object category, $g^{(\ell)} \in \mathcal{G}^{(\ell)}$ are the latent (nuisance) variables at layer ℓ, and $\Lambda^{(\ell)}_{g^{(\ell)}}$ are parameters at layer ℓ. The image I is generated by adding isotropic Gaussian noise η to a now *multi-scale* "rendered" template $\mu_{cg} = \prod_\ell \Lambda^{(\ell)}_{g^{(\ell)}} \mu_{c^{(L)}}$. The template depends on a composition of *nuisance* transformations, one per length scale ℓ. It is useful to define the auxiliary latent variables

$$z^{(\ell)}_n \equiv \Lambda^{(\ell+1)}_{g^{(\ell+1)}_n} \cdots \Lambda^{(L)}_{g^{(L)}_n} \mu_{c^{(L)}_n} \in \mathbb{R}^{D^{(\ell)}} \quad \forall \ell$$

which we refer to as the *intermediate rendered templates*. Given this definition, we can rewrite the DRMM in a recursive hierarchical form:

$$z^{(\ell)}_n = \Lambda_{g^{(\ell+1)}_n} z^{(\ell+1)}_n \in \mathbb{R}^{D^{(\ell)}}$$

Figure 2B-C show a depiction of the DRMM graphical model. Due to the hierarchical form, each transformation Λ_{g^ℓ} partially renders finer-scale details, as we move from abstract to concrete. Note that the factorized structure of μ_{cg} results in an *exponential reduction in the number of free parameters*, from $C \cdot \prod_{\ell=1}^{L} G^{(\ell)} \cdot D^{(0)}$ to $C \cdot D^{L+1} + \sum_{\ell=0}^{L} G^{(\ell)} D^{(\ell-1)} D^{(\ell)}$, where $C \equiv |\mathcal{C}|$ and $G^{(\ell)} \equiv |\mathcal{G}^{(\ell)}|$. This enables efficient inference, learning, and better generalization.

Next we introduce a variant of the DRMM called the *Non-Negative Deep Rendering Mixture Model (NN-DRMM)*, where the intermediate rendered templates are constrained to be non-negative:

$$z^{(\ell)}_n \geq 0 \quad \forall \ell \in \{1, \dots, L\}.$$

This variant is important as it has been proven that the MPC inference in the NN-DRMM (via an efficient dynamic programming algorithm) leads to feedforward propagation in a DCN. The non-negativity constraint seems to be a necessary assumption for that equivalence to hold. Throughout the rest of the report, we use the NN-DRMM for all experiments (see Non-negativity constraint optimization for details on the optimization). For brevity, we drop the prefix NN.

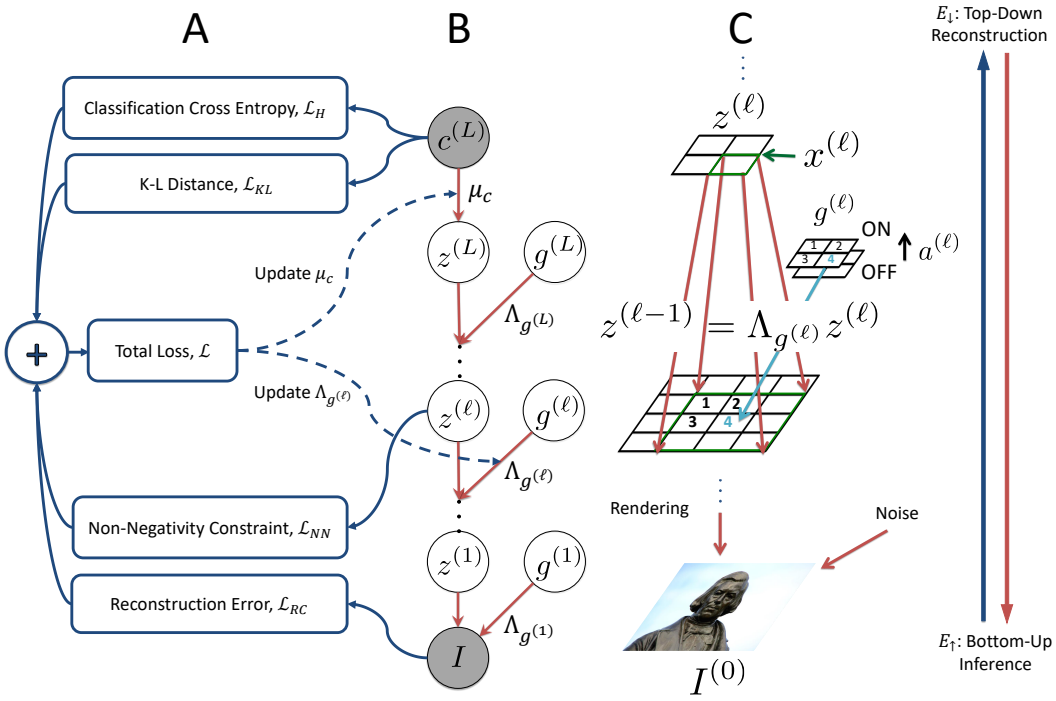


Fig. 2. Semi-supervised learning for Deep Rendering Mixture Model (DRMM). (A) Computation flow for semi-supervised DRMM loss function and its components. Dashed arrows indicate parameter update. (B) The Deep Rendering Mixture Model (DRMM). All dependence on pixel location x has been suppressed for clarity. (C) DRMM generative model: a single super pixel $x^{(\ell)}$ at level ℓ (green, upper) renders down to a 3×3 image patch at level $\ell - 1$ (green, lower), whose location is specified by $g^{(\ell)}$ (light blue). (C) shows only the transformation from level ℓ of the hierarchy of abstraction to level $\ell - 1$.

The MPC bottom-up inference pass in the DRMM is then followed by a top-down inference which uses $(\hat{c}, \hat{g})$ to reconstruct the image $\hat{I} \equiv \mu_{\hat{c}\hat{g}}$ and compute the reconstruction error $\mathcal{L}_{RC} \equiv \frac{1}{N}\sum_{n=1}^{N} \left\| I_n - \hat{I}_n \right\|_2^2$ for the learning objective. It is known that when applying a hard EM algorithm on GMMs, the reconstruction error averaged over the dataset is proportional to the expected complete-data log-likelihood.

Variational Inference Despite its power, the DRMM inference algorithm used thus far ignores uncertainty in the latent nuisance posterior $p(g|I)$ because of the max-marginalization over g in the MPC inference algorithm. We would like to properly account for this uncertainty for two main reasons: (i) our fundamental hypothesis is that the brain performs probabilistic inference and (ii) it is very important for good generalization in the semi-supervised setting to account for uncertainty, since we have very little labeled data.

One approach is to try and compute the true class posterior $p(c|I)$ for the DRMM. But due to the non-standard nature of our MPC inference algorithm, we exact inference might not be feasible, as in many probabilistic models. Instead, we use *variational inference*, a technique that enables the approximate inference of the latent posterior. Mathematically, for the DRMM this means approximating the true class posterior $p(c|I) \approx q(c|I)$, with an approximate posterior q from a tractable family of distributions (e.g. Gaussian or Categorical). In light of the efficient MPC inference algorithm for the DRMM, we strategically choose the tractable family to be $q(c|I) \equiv p(c|\hat{g}, I)$, where $\hat{g} \equiv \operatorname{argmax}_g p(c, g|I)$. In other words, we choose q to be restricted to the DRMM family of nuisance max-marginalized class posteriors. Note that this is indeed an approximation, since the true DRMM class posterior has nuisances that are *sum*-marginalized out $p(c|I) = \sum_g p(c, g|I)$, whereas the approximating variational family has nuisances that are *max*-marginalized out.

Given our choice of variational family q, we derive the variational term for the loss function, starting from the principled goal of minimizing the KL-distance $D_{KL}[q(c|I)||p(c|I)]$ between the true and approximate posteriors with respect to the parameters of q. As a result, such an optimized q will tilt towards better approximating $p(c|I)$, which in turn means that it will account for *some* of the uncertainty in $p(g|I)$. The variational terms in the loss are defined as [17]:

$$\begin{aligned}\mathcal{L}_{VI} &\equiv \mathcal{L}_{RC} + \beta_{KL}\mathcal{L}_{KL} \\ &\equiv -\mathbb{E}_q\left[\ln p(I|c)\right] + \beta_{KL} D_{KL}[q(c|I)||p(c)].\end{aligned}$$

This term is quite similar to that used in variational autoencoders (VAE) [18], except for two key differences: (i) here the latent variable c is discrete categorical rather than continuous Gaussian and (ii) we have employed a slight relaxation of the VAE by allowing for a penalty parameter $\beta_{KL} \neq 1$ motivated by recent experimental results showing that such freedom enables optimal disentangling of the true intrinsic latent variables from the data [19].

The KL divergence $\mathcal{L}_{KL}$ and the expected reconstruction error $\mathcal{L}_{RC}$ are exactly the terms defined earlier for the DRMM's semi-supervised learning loss function. Note that the expected reconstruction error can be computed exactly since (i) the class c is discrete and (ii) our choice of variational family $q(c|I) = p(c|\hat{g}, I)$ only requires that we sample over c and,

importantly, not g. Therefore we can reconstruct I given a sample of c and a fixed inferred $\hat{g}$.

In summary, by optimizing the lower bound of the true class posterior $p(c|I)$ during learning, variational inference allows the DRMM to account for uncertainty in the latent nuisance posterior $p(g|I)$. In other word, the model updates its weights less when the model is not certain about its decision. This behavior results in better performance when the number of labeled data is small and the uncertainty in decision is high.

3. Cortically Inspired Models

Overview of Neural Features for Machine Learning.

Our goal is to build a class of models that use functional elements and computational principles of the cortex for more robust and versatile machine learning. The submitted algorithm for Phase I incorporates three major features that derive from our theoretical framework that the brain is maintaining a generative model of the world, the computational neural model in our bottom-up approach, and the neuro-scientific literature.

Unsupervised and Semi-supervised Learning. Current state-of-the-art machine learning algorithms are trained on enormous sets of labeled examples presented in batches to the learning algorithm. In contrast to that, humans and animals learn from few instances of only weakly labeled examples. When we learned what a cow is, we did not see thousands of different images of a cow, but instead saw maybe a few cows from a few viewpoints. It is widely believed that the brain is building a rich model for the data in an unsupervised way such that novel categories can be constructed quickly. As described above, we built a theory for unsupervised training in neural networks using the framework of generative models. We demonstrate that this allows us to derive a nonlinear Hebbian-type plasticity rule that taps into the large available pool of unlabeled data (unsupervised learning) and learn more from fewer labelled examples (semi-supervised learning), achieving state-of-the-art performance in semi-supervised benchmark tasks.

Representation Principles and Nonlinearities. Our initial DRMM was reverse engineered to find a generative model for standard convolutional neural networks whose non-linear operations and stimulus representations are far simpler than those of the brain. We inferred new principles of stimulus representation and transformation by predicting the responses of neurons recorded in mouse V1 to natural images using our neuro-realistic computational model (NetGard). This identified divisive normalization, a well known mechanism of neural representations, as a major functional element. Divisive normalization is a canonical computation that arises as a natural consequence of many statistical and information theoretic principles on natural signals. We show that building this operation into machine learning yields a distinct advantage in performance, learning rate and learning stability particularly when learning online or with few labeled data, which is much more like the brain's natural environment. Online learning is a critical component for generalized AI, and from a practical point of view it will become especially useful in Phase II segmentation tasks, where memory constraints often force algorithms to learn on smaller batch sizes. Finally, we emphasize that that our computational neural model (NetGard) captures more interesting representational features of the cortex than just divisive normalization (see computational model report for details). This is why we also explored other representational principles such as sparseness and clustering of the population activity in machine learning. As they have not made it into our generative model framework yet, we only briefly mention these informative machine learning experiments in the appendix.

Pruning. The human brain has about 100,000 billion synapses. Given that we only live for about 2.5 billion seconds this is an immensely complex model for such little training data. Current state-of-the-art learning algorithms work in a similar regime. They are usually trained with hundreds of thousands of training points while having on the order of millions parameters. This means that learning complex models need mechanisms to reduce model complexity. Pruning neurons and synapses is one way to do so, which reduces the risk of overfitting, makes models more efficient, and is energetically beneficial for the brain. Our theoretical framework of generative models allows us to formulate new pruning mechanisms that improve the efficiency of our model.

In the sections that follow, we describe in more detail how these three neural features impact machine learning. Encouraged by the successes of our neurally inspired DRMM network (DeepCINN), we have chosen it as our deliverable to be evaluated in the Phase I demonstration task. This particular algorithm is the synthesis of a lot of hard work and the result of a tight collaboration across all the research groups of our team, especially the machine learning members from Toronto and Houston. The resultant Phase I product is a generative model that incorporates the neurally-inspired components of semisupervised learning, divisive normalization, and pruning.

(A) Unsupervised / semi-supervised learning.

Equipped with the DRMM generative model, we can now do supervised, unsupervised and (perhaps most importantly) *semi-supervised* learning in a unified framework (see Fig. 2A-B). Specifically, this allows our networks to learn the rich statistical structure of the inputs while also simultaneously learning the task, to map inputs to desired outputs. By modeling the structure of the input as well as the input-output map, the precious labels can be used to greater advantage by apportioning credit amongst task-relevant input features, while avoiding assigning credit to spuriously correlated features that don't occur in the input.

Early on in the learning process, when the learned model is poor, there will typically be significant uncertainty about the best interpretation for each input. The uncertainty is even greater in semi-supervised setting, wherein the data are sparsely labeled, thereby providing far fewer constraints on the model parameters. As a consequence, early inferences about the best explanation or interpretation are likely to be overconfident. A good semi-supervised learning algorithm should not, therefore, assign too much credit to such early inferences. Motivated by these facts, we use Variational Inference (VI) to better account for uncertainty (see Variational Inference for details) in our semi-supervised learning algorithms. This notion is consistent with the extensive experimental evidence showing that uncertainty does indeed guide the actions of animals as well as how they learn [20].

Loss Function for Semi-supervised Learning. The bene-

fits of learning from labeled data with uncertain interpretations accrue by training the network to optimize a loss function that attempts to reconstruct the inputs from their latent causes inferred from MPC and VI. The model is thus forced to learn enough input features to reconstruct inputs, while the task loss emphasizes the more important feature directions for the specific task at hand. In this way the unsupervised and supervised terms of the loss function work together cooperatively. Mathematically, the loss function for semi-supervised learning in the DRMM is given by $\mathcal{L} \equiv \alpha_{CE}\mathcal{L}_{CE} + \alpha_{RC}\mathcal{L}_{RC} + \alpha_{KL}\mathcal{L}_{KL} + \alpha_{NN}\mathcal{L}_{NN}$, where α_{CE}, α_{RC}, α_{KL} and α_{NN} are the weights for the cross-entropy loss $\mathcal{L}_{CE}$, reconstruction loss $\mathcal{L}_{RC}$, variational inference loss $\mathcal{L}_{KL}$, and the non-negativity penalty loss $\mathcal{L}_{NN}$, respectively. The $\{\alpha_j\}$ are estimated via cross-validation. The cross-entropy loss is given as $\mathcal{L}_H \equiv -\frac{1}{|\mathcal{D}_L|}\sum_{n\in\mathcal{D}_L}\sum_{c\in\mathcal{C}}[\hat{c}_n = c_n]\log q\left(c|I_n\right)$ and the other losses are defined as in Section 2. We summarize those individual losses below.

$$\mathcal{L}_H \equiv -\frac{1}{|\mathcal{D}_L|}\sum_{n\in\mathcal{D}_L}\sum_{c\in\mathcal{C}}[\hat{c}_n = c_n]\log q\left(c|I_n\right) \quad [1]$$

$$\mathcal{L}_{RC} \equiv \frac{1}{N}\sum_{n=1}^{N}\left\|I_n - \hat{I}_n\right\|_2^2 \quad [2]$$

$$\mathcal{L}_{KL} \equiv \frac{1}{N}\sum_{n=1}^{N}\sum_{c\in\mathcal{C}} q\left(c|I_n\right)\log\left(\frac{q\left(c|I_n\right)}{p(c)}\right) \quad [3]$$

$$\mathcal{L}_{NN} \equiv \frac{1}{N}\sum_{n=1}^{N}\sum_{\ell=1}^{L}\left\|\max\left\{0, -z_n^{\ell}\right\}\right\|_2^2. \quad [4]$$

Here, $q\left(c|I_n\right)$ is an approximation of the true posterior $p(c|I_n)$. In the context of the DRMM and the DCN, $q\left(c|I_n\right)$ is the class posterior after the SoftMax layer, $p(c)$ is the class prior, $\mathcal{D}_L$ is the set of labeled images, and $[\hat{c}_n = c_n] = 1$ if $\hat{c}_n = c_n$ and 0 otherwise. The $\max\{0,\cdot\}$ operator is applied element-wise and equivalent to the ReLu activation function used in DCNs. Figure 2A-B shows how the individual losses are computed and combined in our unified framework.

Benefits of Semi-Supervised Learning for Machine Learning. We evaluate our methods on the MNIST, SVHN, CIFAR10, and our synthetic BlenderRender datasets. In all experiments, we perform semi-supervised learning using the DRMM with the training objective including the cross-entropy cost, the reconstruction cost, the KL-distance, and the non-negativity penalties. We train the model on all provided training examples without data augmentation, and vary the number of labels. We report state-of-the-art test errors on MNIST and SVHN. Our results on CIFAR10 are comparable to state-of-the-art methods.* We also show that our DRMM-based semi-supervised learning method outperforms the supervised learning using ConvNets on our BlenderRender dataset (Similarity benchmark) when just a small fraction of the training data have associated labels. More model and training details are provided in the Appendix.

For our first experiments we use the MNIST dataset, which contains 60,000 training images and 10,000 test images of handwritten digits from 0 to 9. Each image is 28-by-28 pixels. To evaluate semi-supervised performance, we randomly choose $N_L \in \{50, 100, 1000\}$ images with labels from the training set such that the amounts of labeled training images from each class are balanced. The remaining training images are provided without labels. We use a 5-layer DRMM with the feedforward configuration similar to the Conv Small network in [21]. We apply batch normalization on the net inputs and use stochastic gradient descent with an exponentially-decayed learning rate to train the model.

Table 1 shows the test error for each experiment. The variational inference (VI) and non-negativity constraint (NN) help improve the semi-supervised learning performance across all setups. In particular, VI alone reduces the test error from 13.41% to 1.36% when $N_L = 100$. Using both VI and NN, the test error is reduced to 0.57%, and the DRMM achieves state-of-the-art results in all experiments.† Notice that the VI alone leads to significant improvements in the test errors. This improvement is significantly larger when the number of labeled data is small ($N_L \in \{50, 100\}$). When the number of labeled data increases, the reduction in test error is decreased (1.64% reduction in test error when $N_L = 1000$ vs. 12.05% reduction in test error when $N_L = 100$). This discrepancy is due to the fact that when there are only a few labeled data, the uncertainty in the latent nuisance posterior $p(c, g|I)$ is high and cannot be ignored. As a result, VI yields significant improvement over MPC (see also Section 2). When there are more labeled data, the uncertainty is reduced, and VI is not as beneficial.

The SVHN dataset contains 73,257 32x32x3 color images of street-view house number digits. For training, we use a 9-layer DRMM with feedforward propagation similar to the Conv Large network in [21]. The remaining training details are the same as those of the MNIST model. We train our model using $N_L = 500, 1K$ labels, and show state-of-the-art results in Table 2.

We use the CIFAR10 dataset to test the semi-supervised learning performance of the DRMM on natural images. For CIFAR10 training, we use the same 9-layer DRMM as for SVHN. Table 3 presents the semi-supervised learning results for the 9-layer DRMM for $N_L = 4K, 8K$ images. Even though we only use a simple stochastic gradient descent algorithm to train our model, the DRMM achieves comparable results to state-of-the-art methods (21.8% versus 20.40% test error when $N_L = 4K$ as with the Ladder Networks). For semi-supervised learning tasks on CIFAR10, the Improved GAN has the best classification error (18.63% and 17.72% test errors when $N_L \in \{4K, 8K\}$). However, unlike the Ladder Networks and the DRMM, GAN-based architectures have an entirely different objective function, approximating the Nash equilibrium of a two-layer minimax game. As such, a better comparison for future work might be a DRMM-GAN: a DRMM architecture with a GAN loss function.

In addition to the popular benchmarks above, we also validate our semi-supervised learning algorithm on our own dataset using our BlenderRender tool (see Similarity benchmark). BlenderRender is an easy-to-use python API built on top of Blender, an open-source rendering engine. BlenderRender enables easy generation of synthetic images and allows the usage of external models such as the ShapeNet library [27]. We use a total of 1,085 models (classes) from 55 categories (superclasses) from ShapeNet to render 110K images

*Note that we only compare our methods with others that do not use data augmentation.

†The results for Improved GAN are on the permutation-invariant MNIST task, while the DRMM performance is on the regular MNIST task. Since the DRMM contains local latent variables t and a at each level, it is not suitable for tasks such as permutation invariant MNIST

Table 1. Test error for semi-supervised learning on MNIST using $N_U = 60K$ unlabeled and $N_L \in \{100, 600, 1K\}$ labeled images.

Model	Test error (%) for a given number of labeled examples		
	$N_L = 50$	$N_L = 100$	$N_L = 1K$
DGN [22]	-	3.33 ± 0.14	2.40 ± 0.02
catGAN [23]	-	1.39 ± 0.28	-
Virtual Adversarial [24]	-	2.12	-
Skip Deep Generative Model [25]	-	1.32	-
LadderNetwork [21]	-	1.06 ± 0.37	0.84 ± 0.08
Auxiliary Deep Generative Model [25]	-	0.96	-
ImprovedGAN [26]	2.21 ± 1.36	0.93 ± 0.065	-
DRMM 5-layer Sup	-	22.98	6.45
DRMM 5-layer + MPC	21.73	13.41	2.35
NN-DRMM 5-layer + MPC	22.10	12.28	2.26
DRMM 5-layer + VI	2.46	1.36	0.71
NN-DRMM 5-layer + VI	**0.91**	**0.57**	**0.6**

Table 2. Test error for semi-supervised learning on SVHN using $N_U = 73,257$ unlabeled and $N_L \in \{500, 1K\}$ labeled images.

Model	Test error (%)	
	500	1000
DGN [22]	-	36.02 ± 0.10
Virtual Adversarial [24]	-	24.63
Auxiliary Deep Generative Model [25]	-	22.86
Skip Deep Generative Model [25]	-	16.61 ± 0.24
ImprovedGAN [26]	18.44 ± 4.8	8.11 ± 1.3
DRMM 9-layer + VI	11.11	9.75
NN-DRMM 9-layer + VI	**9.85**	**6.78**

Table 3. Test error for semi-supervised learning on CIFAR10 using $N_U = 50K$ unlabeled and $N_L \in \{4K, 8K\}$ labeled images.

Model	Test error (%)	
	4000	8000
Ladder network [21]	20.40 ± 0.47	-
CatGAN [23]	19.58 ± 0.46	-
ImprovedGAN [26]	$\mathbf{18.63 \pm 2.32}$	17.72 ± 1.82
DRMM 9-layer Sup	23.33	-
DRMM 9-layer + VI	23.24	20.95
NN-DRMM 9-layer + VI	21.50	**17.16**

with different textures and nuisance configurations such as object orientations and lighting conditions.We will submit the training and the test set along with our final algorithm.

We train a 9-layer DRMM on the BlenderRender dataset for object recognition and shape recognition task in a semi-supervised learning where only 10% of the data is labeled. We observe an advantage of our semi-supervised learning method over supervised learning with an equivalent 9-layer ConvNet. In particular, the 9-layer DRMM trained semi-supervised reduces the test error by 6.02% (65.43% vs. 71.43% test error) and 3.5% (33.67% vs. 37.17%) on the object superclass and class recognition tasks, respectively.

(B) Divisive Normalization.

Divisive normalization is a phenomenological model describing the response behavior of populations of neurons to changes in the contrast of the signal [28, 29]. The core element of divisive normalization is the normalization of the response of a summation field $\mathcal{A}$ by activity in a suppression field $\mathcal{B}$ [2, 28]:

$$\tilde{z}_j = \gamma \frac{\sum_{i \in \mathcal{A}_j} u_i z_i}{\left(\sigma^2 + \sum_{k \in \mathcal{B}_j} w_k z_k^p\right)^{\frac{1}{p}}}, \qquad [5]$$

where $\{u_i\}$ are the summation weights, $\{w_k\}$ the suppression weights, and z_i represent some form of pre-normalized neural activity. The term σ^2 is a scalar semi-saturation constant, influencing the location of the contrast response curve on the contrast axis. The terms γ and p are scalar hyper-parameters. Usually, there is only one neuron in the summation field, while the suppression field $\mathcal{B}$ includes neighboring neurons with different tunings [28, 30]. In this form, the surround contrast multiplicatively modulates the orientation tuning curve of a neuron. Divisive normalization is ubiquitous in the mammalian nervous system and has thus been called a "canonical computation" [2]. Previous theoretical studies have outlined several potential computational roles for divisive normalization such as sensitivity maximization [2], invariant coding [31], density modelling [32], image compression [33], distributed neural representations [34], stimulus decoding [35, 36], winner-take-all mechanisms [37], attention [38], redundancy reduction [39–42], marginalization in neural probabilistic population codes [43], and contextual modulations in neural populations and perception [44, 45].

Although our current computational model NetGard is only fitted on static natural images and the integrated neural activity over 500ms of stimulus presentation, and even though we did not put in the explicit normalization equation into the model, it still learned dynamics that exhibit divisive normalization type transformations (see accompanying Computational Model report). This suggests that this prominent operation plays a significant role in cortical information processing. Therefore, divisive normalization was the obvious candidate to investigate for transferring a canonical operation to machine learning. So far, with a few exceptions [46, 47], divisive normalization has received little attention in conventional supervised deep learning.

On a very fundamental level, the brain is a recurrent system with non-trivial temporal dynamics. As such, it needs to avoid the explosion or vanishing of activities which would either drive neurons into saturation or shut down the network activity. Machine learning algorithms share this problem too. Vanishing or exploding activity limits the timescale at

which recurrent neural networks can be trained. In supervised learning within feedforward networks, activities vary over orders of magnitude, which can be harmful as well because it complicates learning. In AlexNet [47], for instance, the intermediate activations can differ by several orders of magnitude. This can lead to exploding or vanishing gradients, since the Jacobian gets multiplied by the input activation of every layer. Normalization of activations is one way to address this problem.

In machine learning, normalization techniques have been studied primarily in the context of unsupervised learning, and have only recently begun to be explored in supervised learning. Batch normalization (BN) exploits mini-batch statistics to normalize activations [48], which was shown to speed up training and lead to better test results. However its success has so far been limited when applied to recurrent neural networks [49]. Layer normalization (LN), on the other hand, normalizes the activations across all activities within a network layer [50]. This was shown to work well for recurrent networks, but performed suboptimally for convolutional neural networks. We cast all these normalization techniques into a unified framework akin to divisive normalization [28] and explored their efficacy in deep and recurrent neural networks [51]. Compared to batch normalization, divisive normalization (DN) has the distinct advantage that it can be used on single images, making it possible to apply it in online learning as well as recurrent networks. We found that two hallmarks of transformations and representations in cortical networks, the constant σ^2 in the denominator of divisive normalization and a sparse regularizer on neural activations, leads to significant benefits over conventional normalization techniques (see Divisive Normalization for Deep and Recurrent Networks for details). In brief, we found that divisive normalization with sparse regularization outperforms batch normalization in the same convolutional network architecture on the CIFAR classification task, improves the performance on a recurrent neural network word modelling task, and outperforms state-of-the-art networks on a super-resolution task. We also found that we were able use much higher learning rates in recurrent networks, suggesting that divisive normalization makes learning more stable. In summary, we demonstrated that a canonical neural operation can be successfully implemented in current state-of-the-art machine learning models and can improve the performance compared to baseline models.[‡]

Divisive Normalization in the DRMM Given the success of DN in supervised learning and the canonical nature of the operation in cortex, a natural question to ask is: Can DN be incorporated into the DRMM generative framework? We show here that the answer is yes: we can interpret DN as inference in a particular variant of a Gaussian Scale Mixture Model (GSM), as described below.

As a consequence of this theoretical insight, we can integrate DN into the larger DRMM generative modeling framework and, perhaps most importantly, we can perform un-/semi-supervised learning with DN in a principled way. We next describe the mathematics in more detail and report experimental results showing that DN achieves comparable performance to BN in supervised learning and achieves faster, more stable and more accurate performance in semi-supervised learning benchmarks. The latter results shows that DN has significant advantages over BN when labeled data is scarce. We also show that an important utility of divisive normalization in machine learning algorithms is that it enables online training.

From the DRMM inference standpoint, divisive normalization can be interpreted as inferring a latent variable in a Gaussian scale mixture (GSM) [39, 52–54]. More specifically, let the generative model be

$$\begin{aligned}\mathbf{u} &\sim \mathcal{N}(0, I) \in \mathbb{R}^D \\ s &\sim \mathcal{IG}(\alpha, \beta) \in \mathbb{R}_+ \\ \mathbf{z} &\sim \sqrt{s} \cdot \mathbf{u},\end{aligned}$$

where $\mathcal{IG}$ denotes the inverse gamma distribution. Then $\mathbf{z}$ is multivariate t-distributed [55]. Intuitively, $\mathbf{u} \in \mathbb{R}^D$ captures the pattern of an image (patch) while the scalar variable s describes the local contrast. For this generative model, the maximum *a posteriori* estimator (MAP) for $\mathbf{u}$ given $\mathbf{x}$ has the same form as divisive normalization, $\mathbf{u} = \pm\frac{\sqrt{2\beta}\sigma\mathbf{x}}{\sqrt{\alpha\sigma^2 + \|\mathbf{x}\|^2}}$.

In the context of the DRMM and DCNs, $\mathbf{u}$ are the activations and $\mathbf{x}$ are the inputs at each layer in the network.

Benefits of Divisive Normalization for Machine Learning Overall, we find that DN converges as fast or faster, and yields more stable learning curves than BN. Importantly, we find that DN has an especially large advantage in the online learning setting. This is because, in contrast to BN, which requires a batch of images to compute a normalization, DN works with a single image and can thus be in the online setting where one input is processed at a time. Note that online learning is biologically more plausible and applies to more real-world situations than batch learning [§].

We now describe the experiments in more detail. We compare the performance of DN and BN on two distinct tasks (shape/class classification and similarity matching) using two distinct architectures (the 9-layer DRMM described earlier and a VGG-based Convnet). We consider three learning settings: supervised, semi-supervised (only 10% of the labeled data is available), and online (each input is processed one at a time).

(Semi-)Supervised Learning. In our experiments, we train a 9-layer DRMM on the BlenderRender shape classification task in a supervised setting (using all available labeled data) and a semi-supervised setting (using only a small fraction of labeled data). Fig. 4 and Fig. 5 show the test error vs. training epochs for the supervised task (top) and the semi-supervised task (bottom). Overall, DN results in more stable training than BN and this advantage is accentuated in the semi-supervised setting. Also, in the semi-supervised case, DN converges 4x time faster than BN (5 epochs vs. 22 epochs to reach 77% test error).

Online Learning. We compare DN vs. BN in the online learning setting – wherein each input is processed one at a time – for two distinct tasks and architectures. First, we train the 9-layer DRMM architecture supervised on the BlenderRender shape classification task. We find that DN converges faster and also achieves a much better test error compared to

[‡] Note, that on some of the datasets like CIFAR, we do not report state-of-the-art results. To achieve those, many other techniques, like data augmentation, are necessary. To isolate the effect of divisive normalization on the performance, we compare our results to a baseline model that does not use these additional optimizations.

[§] Note that a modified online batch normalization approach could normalize by a weighted moving average, and thereby might interpolate between divisive normalization and batch normalization. We plan to explore this online batch normalization in future work.

the baseline network without normalization (36% vs. 50% test error, see Fig 6). Second, we train the 16-layer VGG network on the BlenderRender similarity task. We find that DN yields a significant advantage when compared to a baseline network without normalization (Fig. 6). In more detail, we evaluate DN (and sparseness, see [56]) in a Siamese network based on the VGG network (see Siamese networks based on VGG). We train this similarity network with and without DN and sparseness on the similarity task in an online setting, presenting one pair of examples at a time. Fig. 3 (top) shows the test accuracy for classifying whether two pairs of images represent the same object while Fig. 3 (bottom) shows 5-way test accuracy (choosing the image most similar to the target image from a set of five reference images). The learning curves show a clear advantage for DN over BN, with an improvement of about 15%.

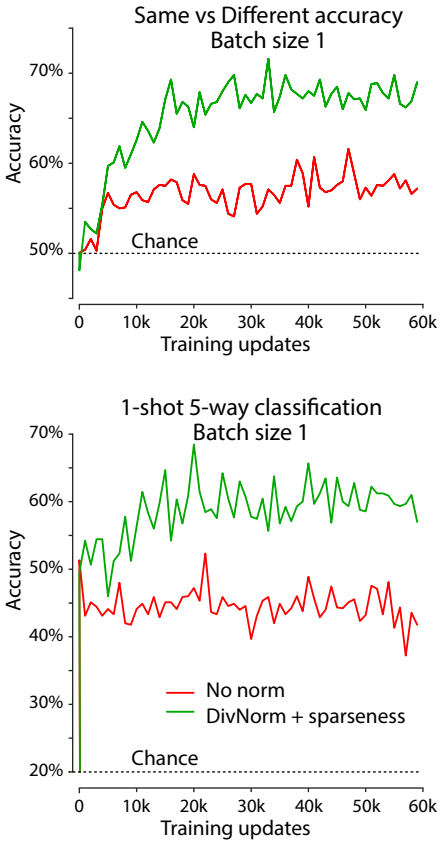


Fig. 3. Benefits of divisive normalization in online learning in similarity discrimination. We trained a VGG based similarity network with and without divisive normalization and sparseness on the similarity task by only presenting one pair of examples at a time. Since batch normalization is ill-defined for that case, we compared the training to a plain Siamese network without any normalization (no norm). The learning curves clearly show an advantage of divisive normalization and sparseness in the online learning setting, reaching around 15% greater accuracy. *Top*: Accuracy in classifying whether a pair of images represent the same object. *Bottom*: Accuracy on the 5-way similarity benchmark. The network must select the target image that depicts the same shape as a given reference from a set of five choices.

(C) Synaptic and Neuron pruning in the DRMM.

In the early years of a child's development, the brain prunes synapses and neurons at a rapid rate [57]. One type of pruning — called *small-scale axon terminal arbor pruning* — is thought

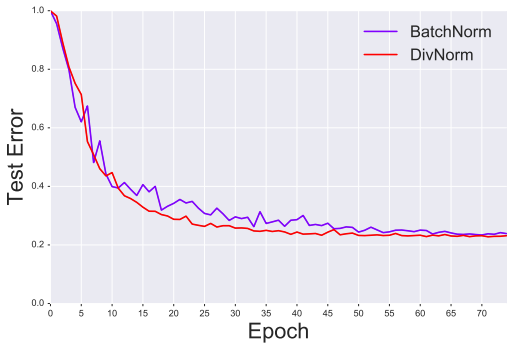


Fig. 4. Comparison between the performance of divisive and batch normalization for the supervised learning task on the BlenderRender benchmark.

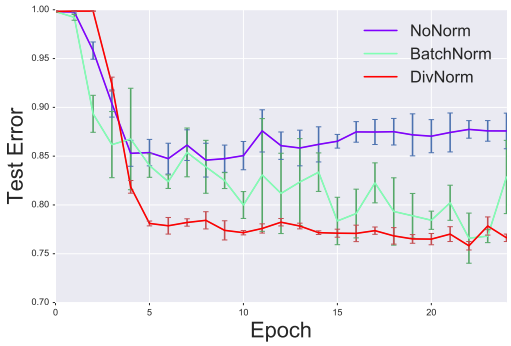


Fig. 5. Comparison between the performance of divisive normalization, batch normalization, and no normalization for the semi-supervised learning task on the BlenderRender benchmark using 10% labeled data. Errors bars determined from averaging over 3 trials.

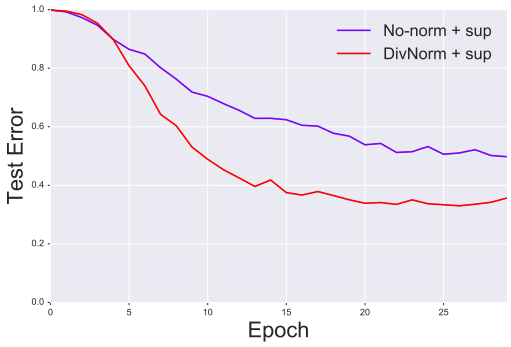


Fig. 6. Comparison between the performance of divisive normalization and no normalization for the online supervised learning task on the BlenderRender benchmark.

to follow a "use-it-or-lose-it" (UILI) rule: synapses that are used regularly are kept while the rest are pruned away [58]. In one brain area, the overall effect of pruning is estimated to be approximately a 41% reduction in the number of neurons from birth to sexual maturation. The mechanisms involved are complex, but the basic premise is simple and is summarized in the UILI mantra. Inspired by this phenomenon and motivated by the need for efficient model selection techniques, we derive novel UILI pruning algorithms in the DRMM framework.

Recall that in the DRMM each switching variable $a_x^\ell \in \{0,1\}$ at position x in level ℓ encodes whether a given dictionary element Λ_{g^ℓ} is used in the rendering of an input. The parameter $\pi_{x,ON}^\ell \equiv p(a_x^\ell = 1)$ controls the prior probability of this latent variable. During the EM-based learning algorithm,

we estimate the π parameters via the simple update rule

$$\hat{\pi}^{\ell}_{x,ON} = \frac{1}{N} \sum_{n} \hat{a}^{\ell}_{nx},$$

where N is the size of the minibatch and $\hat{a}^{\ell}_{nx} \in \{0, 1\}$ is the inferred estimate, *i.e.* the ReLu firing state of that neuron on input I_n. A simple statistical hypothesis test can be used to distinguish between the null and experimental hypotheses $\mathcal{H}_0 : \pi^{\ell}_{x,ON} = 0$ and $\mathcal{H}_1 : \pi^{\ell}_{x,ON} > 0$. This amounts to a thresholding of the estimated parameter: if $\hat{\pi}^{\ell}_{x,ON} > \alpha_N$ then we reject the null hypothesis $\mathcal{H}_0$ and so we will keep the neuron (or synapse). Otherwise we must accept the null hypothesis and prune the neuron (or synapse). This completes the DRMM-based derivation of the *UILI Neuron Pruning (UILI-NP)* rule: it amounts to estimating mixing parameters of the DRMM and then applying a simple statistical hypothesis test.

In order to derive a UILI rule for *synaptic* pruning, we can use an analogous approach to that for the neuron pruning rule. We simply place a mixing parameter π^{ℓ}_{xy} on the presence or absence of a given weight $\lambda^{\ell}_{xy} \equiv (\Lambda_{g^{\ell}})_{xy}$. This hyper-prior on the weights controls *the probability* that a given weight is present or absent. In the M-step we will update our estimates of these weight presence parameters and then apply a statistical hypothesis test to decide whether the synapse should be there (using a threshold $\alpha_S > 0$), analogous with the neuron case. This completes the definition of the *UILI Synaptic Pruning (UILI-SP)* algorithm.

Benefits of Synaptic Pruning for Machine Learning. In this section, we show that our DRMM-based synaptic pruning algorithm[¶] dramatically reduces the number of parameters and floating point operations (FLOPs), at only a small cost in accuracy. We demonstrate this for the object recognition task on the MNIST and BlenderRender datasets. Pruning thus results in faster training/inference and potentially allows the model to be deployed on lower power systems (*e.g.* mobile devices). Our deliverable network incorporates *synapse* pruning for the demonstration task.

The value of pruning for dramatically reducing model sizes opens up an intriguing new possibility: we can potentially build and train much deeper models. For example, we plan to apply pruning to NetGard in Phase II (see Future Directions in Computational Report).

Despite these possibilities, we acknowledge that currently we only have preliminary results and that we need to do many more systematic experiments to better understand and exploit pruning, and its inverse, growing, which we have yet to explore.

We tried our synaptic pruning algorithms on the LeNet-5 network trained on MNIST, using all labeled data. We compare the performance with other pruning methods under the same setting (see Section 3 for more details). We use a similar LeNet-5 as in [59], which achieves 0.8% test error on MNIST without pruning. Table 4 compares our methods with state-of-the-art pruning algorithms in terms of test error, compression rate and FLOP reduction rate. After synaptic pruning, the number of parameters in the model is reduced by 12x and the number of FLOPs is reduced by 8x, at a cost of a slightly worse test error.

During synaptic pruning, there is a trade-off between accuracy and model complexity. Figure 7 (Top) shows the accuracy on the MNIST test set as a function of the percentage of connections or neurons removed by pruning (pruning percentage). Notice that for synaptic pruning, we observe a "free lunch" up to 60% pruning, in which the test accuracy stays almost the same while the model size can be reduced by half.

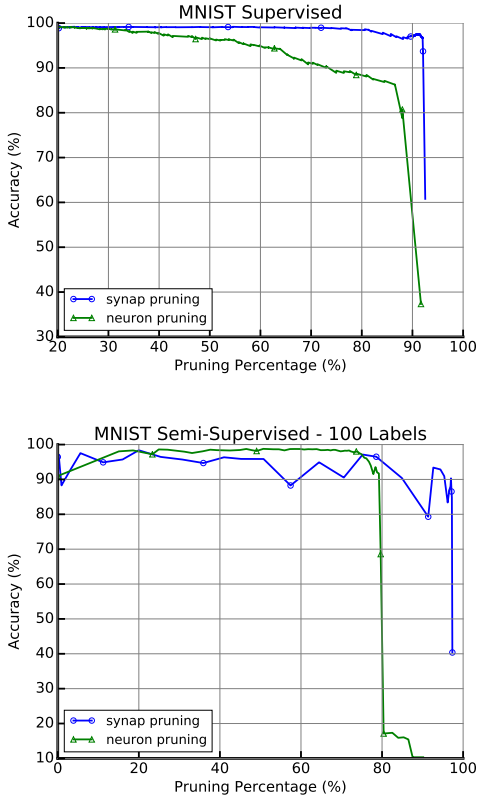


Fig. 7. Test error versus pruning percentage of the DRMMs using synaptic pruning (Blue) and neural pruning (Green). Experiments are done in a supervised setup using 60k labeled MNIST images (top) and in a semi-supervised setup using 60k images with only 100 labels (bottom).

For semi-supervised learning, we use the 5-layer DRMM in [13] and in Section 3 as the baseline. The training setup and hyperparameter values are also the same as in Section 3. We train the model on all provided MNIST training examples using only 100 labeled data points.

Table 5 compares the the semi-supervised learning in the 5-layer DRMM with and without pruning. We also compare our synaptic pruning with the Deep Compression pruning in this setting. The synaptic pruning when applied on the 5-layer DRMM for a semi-supervised learning task achieves comparable test accuracy with the Deep Compression and with the baseline performance without pruning. Our synaptic pruning are outperformed by the Deep Compression pruning in compression rate (1.7x vs. 3.3x) but as good as the Deep Compression pruning in FLOP reduction rate (8.3x). Notice that compared to supervised learning, the number of required FLOPs are almost doubled due to the Top-Down Reconstruction. As a results, the effect of pruning in reducing the number

[¶] We also define and begin to explore the neuron pruning algorithm. However, there are significant complications with the convolutional layers because parameters are shared nonlocally across neurons. For this reason we only show some preliminary results. In the future, when we relax the (neurally implausible) convolutional layers to (neurally plausible) locally connected layers, we will revisit neuron pruning.

Table 4. Comparison with other compression methods on supervised learning task using LeNet-5

Network	Test Error	Parameters	Compress Rate	FLOP Reduction Rate
Baseline LeNet Model [60]	0.8%	431K	1×	1×
Data-Free Pruning [61]	2.01%	78.5K	5×	1.2×
Fastfood-2048 [62]	0.83%	52.1K	8×	
Fastfood-1024 [62]	0.92%	38.8K	11×	
Pruning in Deep Compression [59]	0.77%	36K	12×	6×
Our synaptic pruning	1.24%	35.5K	**12**×	**8**×

of required FLOPs is even more significant, allowing efficient and tractable inference and learning from unlabeled data.

Figure 7 (Bottom) shows the trade-off curve between the semi-supervised learning test error and the pruning percentage for synaptic pruning and neuron pruning.

After preliminary experimentation with our pruning algorithms, it became clear that the choice of pruning hyperparameters is quite important for performance. To resolve this question with biological data, we searched the neuroscience literature for evidence of pruning rate schedules in neocortex and found electron microscopy studies of developing cortex that estimate these schedules from developing rat cortex [63]. Estimates of pruning rates are shown in Table 6, and more details on the pruning schedule can be found in Table 8 in the Appendix.

With these considerations, we applied the rat neocortex-matched pruning rates on a 9-layer DRMM trained via semi-supervised learning for our BlenderRender benchmark using the coarse-scale class labels. The test error and learning rate vs. epochs are shown in Fig. 8. We use this same pruning rate schedule in our submission for the demonstration task. Note however that we have *not* done extensive hyperparameter tuning of the pruning rate schedule; we plan to pursue this more in Phase II along with algorithms for synapse formation.

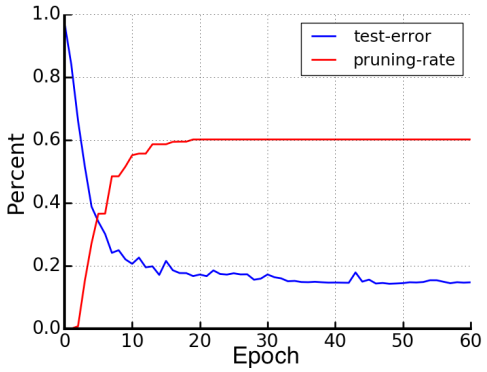


Fig. 8. Test errors and pruning rates during the DRMM pretraining training on the BlenderRender benchmark for the object recognition task.

4. Demonstration task

Similarity judgments refer to the problem of predicting a score for each image in a set of candidate images based on its similarity with a reference image. This problem is especially interesting when the reference and candidate images belong to categories that the model has never seen before. This is related to one-shot learning where a model has to learn a category based on only a single example. The similarity task is challenging for a number of reasons. Firstly, even two images of exactly the same object can appear substantially different because of changes in nuisance variables like the lighting conditions, the pose of the object, its rotation, its texture, and so on. Moreover, without further specification of the dimensions used for judging comparison, similarity judgments are inherently ill-defined [64]. But even after settling on shape as the dimension of interest, the variation in similarity rankings can still be high, forcing us to rely on potentially unreliable similarity labels when training an algorithm in a supervised manner. Finally, another source of difficulty arises from the one-shot learning nature of our evaluation task in which the algorithm has to generalize from a single example.

The machine learning community has been interested in the problem of similarity, mainly in the setting where one class label is available for each image and two images are considered similar if and only if they belong to the same class. The similarity task by nature requires understanding the relationship between different examples instead of understanding something about each example separately. Architectures like Siamese or triplet networks have been proposed, whose inputs are simultaneously-presented tuplets of examples; the networks are trained to predict their relationship. There is a long history of such methods in applications like signature or face verification. Some of the older approaches use shallow embeddings [65, 66] while newer ones exploit deep learning to learn richer representations but with similar loss functions [67, 68]. Siamese networks are also used for one-shot learning of new categories [69] while other approaches are designed specifically for one-shot learning of new categories [70, 71].

Neurally inspired DRMM (DeepCINNs) in Siamese framework.

We use the Siamese network framework to adapt our neurally inspired DRMM (DeepCINN) to the demonstration similarity task. In the car-engine analogy, our neurally inspired DRMM represents the engine and incorporates neurally plausible features and transformations inferred from our data, the literature, and the computational model NetGard, while the Siamese network represents the car that enables us to test the engine in the similarity task (Fig. 9).

Siamese Networks. A Siamese network is comprised of two separate branches with identical weights for the parallel extraction of features for a pair of images. The two networks are trained and their features (usually from the last layers before the Softmax) are combined to provide a nonlinear embedding function into a representation in which the cosine of the angle between the two embeddings yields a faithful similarity metric for comparing objects in the image. An example of a Siamese network is shown in Figure 9. As specified by Sandia, we will use the object's shape as the main

Table 5. Comparison with the pruning method in the Deep Compression on the 5-layer DRMM trained in a semi-supervised setup using 100 labels

Network	Test Error	Parameters	Compress Rate	FLOP Reduction Rate
Baseline 5-layer DRMM [13]	0.57%	131.2K	1×	1×
Pruning in Deep Compression [59]	0.65%	39.36K	**3.3**×	8.3×
Our synaptic pruning	0.78%	77.4K	1.7×	**8.3**×

Table 6. Estimated Pruning Rate Schedule in the Rat Neocortex (adapted from [63])

Postnatal Days	Pruning Rates
19-23	27%
23-27	15%
27-31	3.9%
31-35	0.6%

determinant of similarity.

The problem faced in training Siamese networks on similarity tasks is that we do not have access to ground truth similarity ratings, but only object labels instead. Therefore, we resort to predicting the binary similarity status (similar or not similar) of the objects during training. This can be done by combining the two outputs of the embedding function and using an additional softmax layer on top, which is trained using cross-entropy.

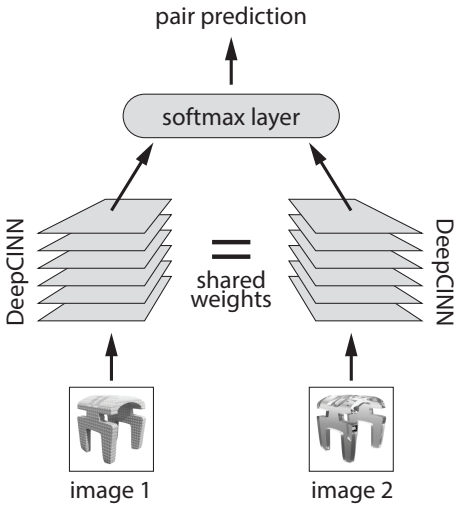


Fig. 9. Siamese network. Two images are fed into identical copies of the same network. The output of the networks are combined to predict the similarity status of that pair.

Architecture. The DeepCINN we use for the submission is a 9-layer DRMM which includes several neurally plausible features: (1) semi-supervised learning with top-down feedback connections and a nonlinear Hebbian learning rule; (2) divisive normalization; and (3) synaptic pruning based on a Use-It-Or-Lose-It rule with a pruning rate schedule derived from biological experiments [63]. The architecture of our DeepCINN is described in Table 7. At each layer, convolution ‖ is followed by divisive normalization and the ReLU activation function. Spatial max-pooling is applied after layers 1, 3, 6 and 9. The Siamese network combines the outputs of layer 9 of the DeepCINN from the two branches by computing their absolute difference.

Training Details. We first pretrain the DeepCINN for the object recognition task on the Tiny ImageNet dataset, and then pretrain it again for the object recognition task on our BlenderRender dataset. Notice that the size of the last layer of the DeepCINN depends on the number of classes in the datasets. In other word, this layer is tailored for the dataset of interest. When we take the DeepCINN trained on the Tiny ImageNet and pretrain it again on the BlenderRender dataset, we replace its last layer by a new one, and then train the network end-to-end. Even though the DeepCINN can learn from just a few labeled examples, in order to get the best features for the similarity learning task later we use all available labeled data, to train the model with our DRMM-based semi-supervised learning algorithm as described in Section 3. Synaptic pruning with the same neocortex-matched pruning rate schedule are applied at both pretraining phases (Table 8). After the pretraining on the Tiny ImageNet, 40% of the network is pruned, and after the pretraining on the BlenderRender dataset, 65% of the network is pruned. Finally, we use the pretrained DeepCINN as the two branches of our Siamese network. We train the Siamese network end-to-end for the similarity learning task by minimizing the binary cross-entropy objective function. The fine-tuning of the DeepCINN Siamese network is done in a supervised manner: since we know the class labels of the images at training time, we can obtain the binary similarity label for each pair (same or different), and compare them to the predicted similarity label obtained by the output of the softmax.

Loss Function. Given P training pairs, and the pair prediction p^i and the ground-truth similarity label y_i for all $i \in 1, \ldots, |P|$, the cross-entropy loss function is:

$$L_H = -\sum_{i=1}^{P} y_i \log p_i + (1 - y_i) \log(1 - p_i) \qquad [6]$$

In order to further improve the performance, after the training converges, we take the trained Siamese network and finetune it again for the same task on the same BlenderRender dataset but with the contrastive regularizer in the objective function. This contrastive regularizer tries to push the images of "positive" pairs closer to each other in embedding space and to separate the images of the "negative" pairs by at least some predefined margin α. Therefore, this loss function operates on the embeddings themselves and is directly concerned with

‖ Full, half, and valid convolutions follow the standards in Theano. Full convolution increases the image size, half convolution reserves the image size, and valid convolution decreases the image size.

their pairwise distances in the learned space. For a pair of images (x_i, x_j):

$$L_{\text{constrast}}(i,j) = \begin{cases} D_{ij}^2 & \text{if positive pair} \\ \max(0, \alpha^2 - D_{ij}^2) & \text{if negative pair} \end{cases} \quad [7]$$

where α is a predefined margin and D_{ij} is the distance between x_i and x_j in an embedding space

$$D_{ij} = \left\| \frac{f(x_i)}{\|f(x_i)\|_2} - \frac{f(x_j)}{\|f(x_j)\|_2} \right\|_2, \quad [8]$$

where $f(x)$ is the embedding of the image x. In our case, the embedding is the outputs of the last layer in the DeepCINN. The final loss is then

$$L_{\text{contrast}} = \sum_{(i,j)\in P} L_{\text{contrast}}(i,j) \quad [9]$$

More details of the training schedule and hyper-parameter values are provided in Table 15 in the Appendix.

Similarity benchmark.

In order to train and test the DeepCINN, we created our own similarity benchmark tasks. We rendered 110K grayscale images from ShapeNet models using our BlenderRender API. ShapeNet [27] is a large-scale dataset of more than 51K three-dimensional shapes. Each shape in ShapeNet belongs to one of 55 categories. For our dataset, we take 1,085 shapes (class) from 55 categories (superclass), and render 100 images for each shape with randomized location, rotation, light location, and texture, using BlenderRender. Shape location is chosen from Top Left, Top Right, Bottom Left, Bottom Right, Center; Light location is chosen from Top Left, Top Right, Bottom Left, Bottom Right, Center; Texture is chosen from 15 variations of Blender built-in procedural textures.

Our BlenderRender test trials (Figure 10) present one reference and five choices (one target and four distractors). These choice sets have shape attributes selected from one of three possible conditions. The conditions are defined by which attributes the choice images share with the reference. The similarity judgment may be based on class, which we define as a specific shape; or superclass, which we define as the same object type but a different shape, like two different kinds of chairs. The three shape conditions are:

1. *Different class*: target has same class as reference, but distractors have other classes.
2. *Different superclass*: target has same class as reference, but distractors have different *super*classes
3. *Same superclass*: target has same *superclass* as reference but different *class*, while distractors have different superclasses

The last condition is the most difficult as it forces the similarity judgment to generalize within a superclass.

None of the shapes and textures that were used for training the models were used for testing its performance. Moreover, to test the performance of our model under even more difficult conditions we created a completely new dataset of stick figures. This dataset contains 20K images of 2K randomly assembled stick shapes. Each shape is rendered 10 times with random rotation and texture. Textures are chosen from a collection of gray pattern images.

Results.

Figure 10 shows performance on the similarity benchmark task of our submitted DeepCINN and a related baseline network.[**] Both networks are based on the DRMM within a Siamese network architecture, with the novel neurally-inspired features of (1) semi-supervised learning, (2) divisive normalization, and (3) synaptic pruning. As we demonstrated in the preceding sections, these neural features have their greatest advantages over baseline models in the online learning setting with little supervision. However, to maximize performance on the demonstration task we trained our final submitted network using all of our training data and computing gradients with large batch sizes as described above. Figure 10 summarizes our results for the submitted network. As a result of using all of the labeled data and large batch sizes, it is not surprising that our DeepCINN performs similarly to the baseline network. Importantly, our DeepCINN performs well above chance even in very challenging cases like our stick figure dataset which is unlike any of its training data.

5. Future directions

As NetGard grows in complexity to improve the match to the functional and structural properties of the canonical cortical circuitry, its underlying model bias for interpreting sense data grows in complexity as well. We are taking several different approaches to understand the principles embodied in this computational circuit model. These methods include reverse-engineering the dynamics equations to produce new generative models and learning rules in the DRM family, and conducting population analyses to reveal the form of message-passing inference.

Reverse engineering NetGard dynamics for a recurrent DRMM

Cortical circuits have extensive lateral connections largely absent from object recognition networks. NetGard directly models these lateral connections, and we will incorporate these modeled structures into the next-generation DRMM. We expect that lateral connections will naturally account for inferential operations associated with dynamic stimuli. Currently, the DRMM has been designed for static images only. As we move to video, we are introducing the *Dynamic* DRMM, in which the latent nuisance variables are endowed with temporal dynamics. We expect that inference in the Dynamic DRMM should lead to a Recurrent Convolutional Net with like-to-like connectivity preferences (similar to V1), as past latent variables predict current ones. Strongest couplings should emerge between nodes encoding latent states that are close to each other according to group transformations like translation or rotation [72, 73].

We will also reverse-engineer natural computational roles of other NetGard cell types. For example, the NetGard equations suggest an inferential role for Martinotti cells (see computational model report). Experimental evidence from neuroanatomy and physiology hints that these cells may mediate some top-down attentional modulation based on the overall brain state of the animal. We plan to explore this further, both experimentally and theoretically. For the latter, we are currently searching for an inferential analog of attention in DRMMs.

[**] Our submitted network structure is identical to that reported here, but actually has been trained slightly longer until the deadline. Performance at the two time points is extremely similar.

Table 7. Details of the DRMM architecture for the competition

DRMM architecture	
Input	4096 (flattened 64x64x1).
E-step Bottom-Up	Conv 96x3x3 (Half), Maxpool 2x2, Conv 96x3x3 (Full), 96x3x3 (Full), Maxpool 2x2, Conv 192x3x3 (Valid), 192x3x3 (Full), 192x3x3 (Valid), Maxpool 2x2, Conv 192x3x3 (Valid), 192x1x1 (Valid), N_classes x1x1 (Valid), Meanpool 6x6, Softmax. DivNorm followed by ReLU activations after each Conv layer
Classes	Object Recognition Pretrainings: N_classes=200 (Tiny ImageNet), N_classes=36 (BlenderRender dataset). Similarity Learning: N_classes=720
E-step Top-Down	DRMM Top-Down Reconstruction.Upsampling nearest-neighbor.

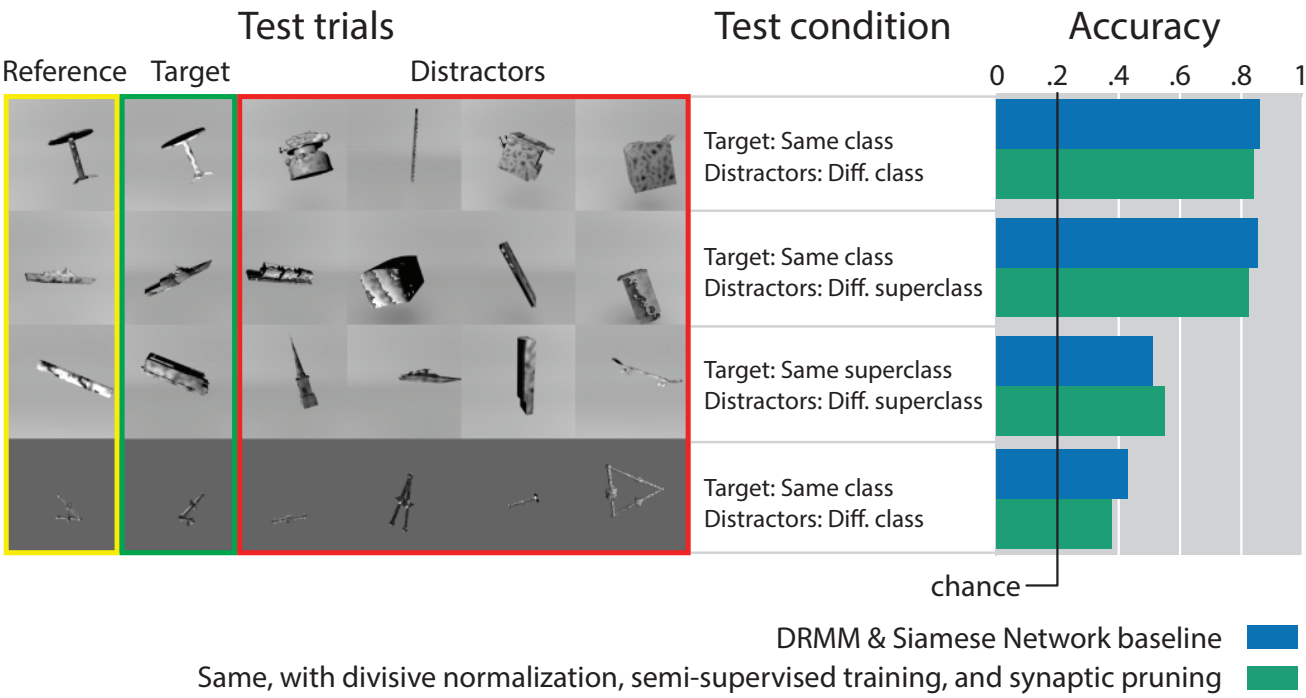


Fig. 10. Performance of submitted networks on our benchmark similarity task. Each row represents a single trial, and one trial is shown for each of four variations of the similarity task we tested. These four conditions test the generalization performance of our network under tasks of varying difficulty (see 4). The first column contains reference images against which similarities are to be judged. The second column contains the target images, which an ideal observer should judge as most similar. The remaining four image columns contain distractors, choices that may share some properties with the reference but should be judged as less similar than the target. The next column specifies the test conditions in each row. The rightmost column quantifies the performance of our submitted deep cortically-inspired neural network (DeepCINN) and a baseline comparison network without the neural features. These are both 9-layer DRMM models used in a Siamese Network configuration. The baseline model is trained with supervised learning on the TinyImageNet database. The model with neural features uses Divisive Normalization and semi-supervised training. Since the target is presented as one of five choices, 0.2 is chance level performance for guessing the target as most similar to the reference. The neurally-inspired DRMM has comparable performance to the baseline model in all conditions.

Table 8. DRMM Pruning Rate Schedule for Semi-Supervised Learning on the Tiny ImageNet and BlenderRender Dataset

Layer	Pruning Rate Schedule	Pruning Period (epochs/prune)
1	27%, 15%, 3.9%, 0.6%	5
2-3	27%, 15%, 3.9%, 0.6%	4
4 -6	27%, 15%, 3.9%, 0.6%	3
7 -9	27%, 15%, 3.9%, 0.6%	2

Additionally, we want to explore the way the inputs are transformed before they enter the cortically inspired NetGard model. Currently, all deep learning architectures for video take in *frames* as input. But all mammalian and insect cortices instead take in *retinal event-driven* input. The power of event-driven processing is already well-known in streaming video compression and in high frequency financial securities trading wherein high throughput and fast reaction times demand that only meaningful changes — events – are processed. Recently, a retinomorphic event-driven video camera called the Dynamic Vision Sensor (DVS) [74] demonstrated high

throughput, dynamic range and low latency in difficult tasks like pencil-balancing. We are building a TensorFlow model of a retina that captures key elements of its processing and representational power. Preliminary results show our DRMM networks exhibit a 10-20% reduction in recognition error on video action recognition benchmarks like KTH [75] when trained with event-based inputs rather than the raw movie. Similarly, NetGard and other neurally-inspired recurrent nets may perform much better when given more biological, event-driven representations of the dynamic inputs.

Natural synaptic plasticity, pruning and growth, and reinforcement learning Due to the lack of evidence for backpropagation in biological systems, in Phase II we will start experimenting with other neuroscience-informed plasticity rules.

A major difference between the neocortex and current machine learning networks is that the cortex includes a greater variety of distinct cell types with that have distinct properties. Our experiments reveal that plasticity rules are likewise cell-type specific (see computational report). We aim to reverse-engineer these rules and their long-term consequences when implemented in NetGard. For example, non-associative plasticity rules may relate to homeostasis, synaptic pruning and growth. In the DRMM framework, such effects correspond to online model selection, and allow model complexity to be adjusted flexibly based on incoming sensory evidence. As another example, we find that associative plasticity between excitatory and inhibitory (PV+) cells changed from anti-Hebbian to Hebbian depending on whether connections were reciprocal or not (see computational model report). We speculate that this selectively maintains reciprocal interactions between inhibitory and excitatory neurons, which may terminate responses to changing latent variables, and allows evidence to shift to nearby latent states according to a DRMM model for dynamic stimuli. Plasticity is also heavily influenced by neuromodulatory inputs such as acetylcholine and dopamine associated with memory and reward. We plan to explore deep architectures that combine unsupervised DRMM-based learning with plasticity modulators based on reinforcement learning.

Neurally-derived message-passing algorithms in machine learning In both NetGard and the brain, information about many variables is distributed across populations of neurons, and is manipulated by the nonlinear transformations that the neurons implement. We hypothesize that these computations implement a message-passing algorithm operating on a probabilistic graphical model whose interactions are encoded by overlapping probabilistic population codes. We are developing an analysis method to reverse-engineer this algorithm from real or synthetic neural data evoked during perceptual inference tasks. The method simultaneously finds (1) the representation of task-relevant variables, (2) interactions between the decoded variables that define the brain's internal model of the world, and (3) the global hyperparameters that define the message-passing inference algorithm. We hypothesize that the global parameters are canonical — that is, common to all parts of the graphical model regardless of interaction strength — so that they generalize to new graphical models. Once we find these nonlinear message-passing operations, we can run them over new generative models such as the DRMM or its generalizations, producing a new recurrent neural network inference algorithm.

By the end of Phase II we plan to incorporate new abstract learning rules that describe how the graphical model changes as a function of the stimulus ensemble. This stimulus ensemble would gradually alter the connectivity patterns according to the synaptic plasticity rules we will incorporate into next-generation NetGard models. The changed circuit connectivity would reshape the population activity dynamics and thus alter the implicit graphical model. By applying our message-passing inference algorithm to the population activity at different times as NetGard learns, we can extrapolate from the short-term, neuro-derived plasticity rules to the long-term learning of a probabilistic graphical model embedded in neural populations.

6. Contributions

Deep Rendering Mixture Model is developed by Ankit Patel, Rich Baraniuk, and Tan Nguyen at Rice and Baylor College of Medicine (BCM) in close interaction with the other team members at Rice and BCM.

Siamese network is developed by Eleni Triantafillou in the labs of Raquel Urtasun and Richard Zemel at the University of Toronto, and was adapted to the DRMM framework by Tan Nguyen, Robin Liu, and Ankit Patel at BCM and Rice.

Divisive normalization and sparseness for deep and recurrent networks has been developed by Mengye Ren and Renjie Liao in the labs of Raquel Urtasun and Richard Zemel at the University of Toronto, in collaboration with Fabian Sinz in the lab of Andreas Tolias at BCM.

Activity clustering has been developed by Renjie Liao, Alexander Schwing (now University of Illinois) in the labs of Raquel Urtasun and Richard Zemel at the University of Toronto.

NetGard is developed by Edgar Y. Walker, Fabian Sinz, and Erick Cobos in the lab of Andreas Tolias at Baylor College of Medicine (BCM), with contributions from other team members at BCM and Rice, as well as the University of Tübingen.

Message passing analysis is developed by Rajkumar Raju and KiJung Yoon in the lab of Xaq Pitkow at BCM and Rice.

Artificial data to train our algorithms are generated by Shenlong Wang in the lab of Raquel Urtasun at University of Toronto, Robin Liu in the labs of Richard Baraniuk and Ankit Patel at Rice (*BlenderRender* benchmark), as well as Thomas Laroche and Edgar Y. Walker in the lab of Andreas Tolias at BCM.

NINAI's effort to advance artificial intelligence by creating novel neuroscience-inspired machine learning algorithms is led by Andreas Tolias (PI) and Xaq Pitkow (co-PI), and coordinated by Jacob Reimer (program and experimental coordinator) and Fabian Sinz (machine learning coordinator).

1. Goodfellow IJ, Shlens J, Szegedy C (2015) Explaining and Harnessing Adversarial Examples in *Iclr 2015*. pp. 1–11.
2. Carandini M, Heeger DJ (2012) Normalization as a canonical neural computation. *Nature reviews. Neuroscience* 13(1):51–62.
3. Ioffe S, Szegedy C (2015) Batch Normalization : Accelerating Deep Network Training by Reducing Internal Covariate Shift, Technical report.
4. Laplace PS (1812) *Théorie Analytique des Probabilités*. (Ve Courcier, Paris).
5. Jaynes ET (2005) Probability theory: the logic of science.
6. Koller D, Friedman N (2009) *Probabilistic graphical models: principles and techniques*. (MIT press).
7. Marr D (1983) *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. (W. H. Freeman), p. 397.
8. Patel AB, Nguyen T, Baraniuk RG (2015) A Probabilistic Theory of Deep Learning, Technical report.

9. Szegedy C, et al. (2013) Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*.
10. Lochmann T, Deneve S (2011) Neural processing as causal inference. *Current opinion in neurobiology* 21(5):774–81.
11. Patel AB, Nguyen T, Baraniuk RG (2015) A probabilistic theory of deep learning. *arXiv preprint arXiv:1504.00641*.
12. Patel AB, Nguyen T, Baraniuk RG (2016) A probabilistic framework for deep learning. *NIPS*.
13. Nguyen T, Patel AB, Baraniuk RG (2017) Semi-supervised learning with deep rendering mixture model. *CVPR(Submitted)*.
14. Ng A, Jordan M (2002) On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. *Advances in neural information processing systems* 14:841.
15. Minka T (2005) Discriminative models, not discriminative training, (Technical Report MSR-TR-2005-144, Microsoft Research), Technical report.
16. Bishop CM, Lasserre J, , et al. (2007) Generative or discriminative? getting the best of both worlds. *Bayesian Statistics* 8:3–24.
17. Blei DM, Kucukelbir A, McAuliffe JD (2016) Variational inference: A review for statisticians. *arXiv preprint arXiv:1601.00670*.
18. Kingma D, Ba J (2014) Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
19. Higgins I, et al. (2016) Early visual concept learning with unsupervised deep learning. *arXiv preprint arXiv:1606.05579*.
20. Doya K, Ishii S, Pouget A, Rao R (2007) *Bayesian brain: Probabilistic approaches to neural coding*. (MIT press).
21. Rasmus A, Berglund M, Honkala M, Valpola H, Raiko T (2015) Semi-supervised learning with ladder networks in *Advances in Neural Information Processing Systems*. pp. 3532–3540.
22. Kingma DP, Mohamed S, Rezende DJ, Welling M (2014) Semi-supervised learning with deep generative models in *Advances in Neural Information Processing Systems*. pp. 3581–3589.
23. Springenberg JT (2015) Unsupervised and semi-supervised learning with categorical generative adversarial networks. *arXiv preprint arXiv:1511.06390*.
24. Miyato T, Maeda Si, Koyama M, Nakae K, Ishii S (2015) Distributional smoothing by virtual adversarial examples. *arXiv preprint arXiv:1507.00677*.
25. Maaløe L, Sønderby CK, Sønderby SK, Winther O (2016) Auxiliary deep generative models. *arXiv preprint arXiv:1602.05473*.
26. Salimans T, et al. (2016) Improved techniques for training gans. *arXiv preprint arXiv:1606.03498*.
27. Chang AX, et al. (2015) ShapeNet: An Information-Rich 3D Model Repository. *ArXiv e-prints*.
28. Heeger DJ (1992) Normalization of cell responses in cat striate cortex. *Vis Neurosci* 9(2):181–197.
29. Albrecht DG, Geisler WS (1991) Motion selectivity and the contrast-response function of simple cells in the visual cortex. *Visual Neuroscience* 7(6):531–546.
30. Bonds AB (1989) Role of Inhibition in the Specification of Orientation Selectivity of Cells in the Cat Striate Cortex. *Visual Neuroscience* 2(01):41–55.
31. Olsen SR, Bhandawat V, Wilson RI (2010) Divisive Normalization in Olfactory Population Codes. *Neuron* 66(2):287–299.
32. Ballé J, Laparra V, Simoncelli EP (2015) Density Modeling of Images using a Generalized Normalization Transformation. pp. 1–12.
33. Malo J, Epifanio I, Navarro R, Simoncelli EP (2006) Nonlinear image representation for efficient perceptual coding. *IEEE Transactions on Image Processing* 15(1):68–80.
34. Simoncelli EP, Heeger DJ (1998) A model of neuronal responses in visual area MT. *Vision Research* 38(5):743–761.
35. Ringach DL (2009) Population coding under normalization. *Vision Research* 50(22):2223–2232.
36. Froudarakis E, et al. (2014) Population code in mouse V1 facilitates readout of natural scenes through increased sparseness. *Nature neuroscience* 17(6):851–7.
37. Busse L, Wade AR, Carandini M (2009) Representation of Concurrent Stimuli by Population Activity in Visual Cortex. *Neuron* 64(6):931–942.
38. Reynolds JH, Heeger DJ (2009) The normalization model of attention. *Neuron* 61(2):168–85.
39. Schwartz O, Simoncelli EP (2001) Natural signal statistics and sensory gain control. *Nat Neurosci* 4(8):819–825.
40. Sinz FH, Bethge M (2008) The Conjoint Effect of Divisive Normalization and Orientation Selectivity on Redundancy Reduction in *Advances in Neural Information Processing Systems 21*, eds. Koller D, Schuurmans D, Bengio Y, Bottou L. (Curran, Red Hook, NY, USA), pp. 1521–1528.
41. Lyu S, Simoncelli EP (2008) Reducing statistical dependencies in natural signals using radial Gaussianization. *Advances in neural information processing systems* 2008:1009–1016.
42. Sinz F, Bethge M (2013) Temporal Adaptation Enhances Efficient Contrast Gain Control on Natural Images. *PLoS Computational Biology* 9(1):e1002889.
43. Beck JM, Latham PE, Pouget A (2011) Marginalization in Neural Circuits with Divisive Normalization. *The Journal of neuroscience : the official journal of the Society for Neuroscience* 31(43):15310–9.
44. Coen-Cagli R, Kohn A, Schwartz O (2015) Flexible gating of contextual influences in natural vision. *Nature Neuroscience* 18(11):1648–1655.
45. Schwartz O, Sejnowski TJ, Dayan P (2009) Perceptual organization in the tilt illusion. *Journal of Vision* 9(4):1–20.
46. Jarrett K, Kavukcuoglu K, Ranzato MA, LeCun Y (2009) What is the best multi-stage architecture for object recognition? *2009 IEEE 12th International Conference on Computer Vision* pp. 2146–2153.
47. Krizhevsky A, Sutskever I, Hinton GE (2012) ImageNet Classification with Deep Convolutional Neural Networks. *Advances In Neural Information Processing Systems* pp. 1–9.
48. Ioffe S, Szegedy C (2015) Batch normalization: Accelerating deep network training by reducing internal covariate shift in *ICML*. pp. 448–456.
49. Laurent C, Pereyra G, Brakel P, Zhang, Y.and Bengio Y (2015) Batch normalized recurrent neural networks. *arXiv preprint arXiv:1510.01378*.
50. Ba JL, Kiros JR, Hinton GE (2016) Layer normalization. *arXiv preprint arXiv:1607.06450*.
51. Ren M, Liao R, Urtasun R, Sinz FH, Zemel RS (2017) Normalizing the normalizers: Comparing and extending network normalization schemes in *submitted to ICLR 2017*. pp. 1–15.
52. Lyu S, Simoncelli EP (2009) Nonlinear extraction of independent components of natural images using radial gaussianization. *Neural Computation* 21(6):1485–1519.
53. Wainwright MJ, Simoncelli EP (2000) Scale mixtures of Gaussians and the statistics of natural images. *Neural Information Processing Systems* 12(1):855–861.
54. Lyu S (2011) Dependency Reduction with Divisive Normalization: Justification and Effectiveness. *Neural Computation* 23(11):2942–2973.
55. Andrews DF, Mallows CL (1974) Scale mixtures of normal distributions. *Journal of the Royal Statistical Society. Series B (Methodological)* pp. 99–102.
56. Ren M, Liao R, Urtasun R, Sinz FH, Zemel RS (2016) Normalizing the normalizers: Comparing and extending network normalization schemes. *arXiv preprint arXiv:1611.04520*.
57. Chechik G, Meilijson I, Ruppin E (1999) Neuronal regulation: A mechanism for synaptic pruning during brain maturation. *Neural computation* 11(8):2061–2080.
58. Allred RP, Kim SY, Jones TA (2014) Use it and/or lose itâexperience effects on brain remodeling across time after stroke. *Frontiers in Human Neuroscience* 8:379.
59. Han S, Mao H, Dally WJ (2015) Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*.
60. LeCun Y, Bottou L, Bengio Y, Haffner P (1998) Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86(11):2278–2324.
61. Srinivas S, Babu RV (2015) Data-free parameter pruning for deep neural networks in *BMVC*.
62. Yang Z, et al. (2015) Deep fried convnets in *2015 IEEE International Conference on Computer Vision (ICCV)*. pp. 1476–1483.
63. Navlakha S, Barth AL, Bar-Joseph Z (2015) Decreasing-rate pruning optimizes the construction of efficient and robust distributed networks. *PLoS Comput Biol* 11(7):e1004347.
64. Medin DL, Goldstone RL, Gentner D (1993) Respects for similarity. *Psychological Review* 100(2):254–278.
65. Bromley J, et al. (1993) Signature verification using a "siamese" time delay neural network. *International Journal of Pattern Recognition and Artificial Intelligence* 7(04):669–688.
66. Weinberger KQ, Blitzer J, Saul LK (2005) Distance metric learning for large margin nearest neighbor classification in *Advances in neural information processing systems*. pp. 1473–1480.
67. Chopra S, Hadsell R, LeCun Y (2005) Learning a similarity metric discriminatively, with application to face verification in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. (IEEE), Vol. 1, pp. 539–546.
68. Schroff F, Kalenichenko D, Philbin J (2015) Facenet: A unified embedding for face recognition and clustering in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. pp. 815–823.
69. Koch G (2015) Ph.D. thesis (University of Toronto).
70. Lake BM, Salakhutdinov R, Tenenbaum JB (2015) Human-level concept learning through probabilistic program induction. *Science* 350(6266):1332–1338.
71. Vinyals O, Blundell C, Lillicrap T, Wierstra D, , et al. (2016) Matching networks for one shot learning in *Advances In Neural Information Processing Systems*. pp. 3630–3638.
72. Wiskott L, Sejnowski TJ (2002) Slow feature analysis: Unsupervised learning of invariances. *Neural computation* 14(4):715–770.
73. Pitkow X, Sompolinsky H, Meister M (2007) A Neural Computation for Visual Acuity in the Presence of Eye Movements. *PLoS Biology* 5(12):e331 EP –.
74. Posch C, Serrano-Gotarredona T, Linares-Barranco B, Delbruck T (2014) Retinomorphic event-based vision sensors: Bioinspired cameras with spiking output. *Proceedings of the IEEE* 102(10):1470–1484.
75. Schuldt C, Laptev I, Caputo B (2004) Recognizing human actions: a local svm approach in *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*. (IEEE), Vol. 3, pp. 32–36.
76. Hinton GE, et al. (2014) Dropout : A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research (JMLR)* 15:1929–1958.
77. Wang Z, Bovik AC, Sheikh HR, Simoncelli EP (2004) Image quality assessment: from error visibility to structural similarity. *IEEE TIP* 13(4):600–612.
78. Zeyde R, Elad M, Protter M (2010) On single image scale-up using sparse-representations in *International conference on curves and surfaces*. (Springer), pp. 711–730.
79. Martin D, Fowlkes C, Tal D, Malik J (2001) A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics in *ICCV*.
80. Timofte R, De Smet V, Van Gool L (2013) Anchored neighborhood regression for fast example-based super-resolution in *ICCV*. pp. 1920–1927.
81. Dong C, Loy CC, He K, Tang X (2016) Image super-resolution using deep convolutional networks. *IEEE TPAMI* 38(2):295–307.
82. De Lathauwer L, De Moor B, Vandewalle J (2000) A multilinear singular value decomposition. *SIAM Journal on Matrix Analysis and Applications*.
83. Krizhevsky A, Hinton G (2009) Learning multiple layers of features from tiny images.
84. Jia Y, et al. (2014) Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*.
85. Cogswell M, Ahmed F, Girshick R, Zitnick L, Batra D (2016) Reducing Overfitting in Deep Networks by Decorrelating Representations. *Proc. ICLR*.
86. Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R (2014) Dropout: A simple way to prevent neural networks from overfitting. *JMLR*.
87. Tieleman T, Hinton G (2012) Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning* 4(2).

7. Appendix

Non-negativity constraint optimization.

In order to derive the DCNs from the DRMM, the intermediate rendered templates $z_n^{(\ell)}$ must be non-negative [12]. This is necessary in order to apply the max-product algorithm, wherein we can push the max to the right to get an efficient message passing algorithm. We enforce this condition in the top-down inference of the DRMM by introducing new non-negativity constrains $z_n^{(\ell)} \geq 0$, $\forall \ell \in \{1, \ldots, L\}$ into the learning objective. There are various well-developed methods to solve optimization problems with non-negativity constraints exactly. For simplicity, in the DRMM learning, we employ an approximate penalty-based approach, which adds an extra non-negativity penalty, in this case $\mathcal{L}_{NN} \equiv \frac{1}{N}\sum_{n=1}^{N}\sum_{\ell=1}^{L}\left\|\max\left\{0, -z_n^{\ell}\right\}\right\|_2^2$, into the objective function. This yields an unconstrained optimization which can be solved by gradient-based methods such as stochastic gradient descent. We cross-validate the penalty weight α_{NN}. On various semi-supervised learning benchmarks, we observe that encouraging this non-negativity constraint on $z^{(\ell)}$ through the loss $\mathcal{L}_{NN}$ yields better performance than truncating negative $z^{(\ell)}$ in the top-down reconstruction, an ad-hoc but widely used technique.

Divisive Normalization for Deep and Recurrent Networks.

Without loss of generality, we denote the hidden input activation of one arbitrary layer in a deep neural network as $\mathbf{z} \in \mathbb{R}^{N \times L}$ where N is the mini-batch size. In the case of a convolutional neural network (CNN), $L = H \times W \times C$, where H, W are the height and width of the convolutional feature map and C is the number of filters or channels. For an RNN or fully-connected layers of a neural net, L is the number of hidden units.

Different normalization methods gather statistics from different ranges of the tensor and then perform normalization. Consider the following general form:

$$z_{n,j} = \sum_i w_{i,j} x_{n,i} + b_j \quad [10]$$

$$v_{n,j} = z_{n,j} - \mathbb{E}_{\mathcal{A}_{n,j}}[z] \quad [11]$$

$$\tilde{z}_{n,j} = \frac{v_{n,j}}{\sqrt{\sigma^2 + \mathbb{E}_{\mathcal{B}_{n,j}}[v^2]}} \quad [12]$$

where $\mathcal{A}_j$ and $\mathcal{B}_j$ are subsets of z and v respectively. One can cast each normalization scheme into this general formulation, where the schemes vary based on how they define these two fields. These definitions are specified in Table 9.

We propose DN as a new local normalization scheme in neural networks. In convolutional layers, it operates on a local spatial window across filter channels, and in fully connected layers it operates on a slice of a hidden state vector.

One obvious way in which the normalization schemes differ is in terms of the information that they combine for normalizing the activations: BN averages over different examples in a mini-batch, but does not pool over features and space, LN pools over all spatial dimensions, but not over features and examples, while DN pools over a small set of features and space, but not over examples.

A second more subtle but important difference between standard BN and LN as opposed to DN is the smoothing term σ, in the denominator of Eq. (12). This term allows some control of the bias of the variance estimation, effectively smoothing the estimate. This is beneficial because divisive normalization does not utilize information from the mini-batch as in BN, and combines information from a smaller field than LN.

While integrating divisive normalization into deep networks, we noticed that normalization schemes could lead to quite correlated filter responses on lower layers (Fig. 11, compare "Baseline" against columns without "*"). We suspected that this might impair performance, because the estimate of the variance in the normalizer suffers when using dependent data points. Empirically, we found that putting a sparse (L1) regularizer

$$\mathcal{L}_{L1} = \alpha \frac{1}{NL} \sum_{n,j} |v_{n,j}| \quad [13]$$

on the centered activations $v_{n,j}$ helps decorrelating the filter responses (Fig. 11, columns with "*"). Here, N is the batch size and L is the number of hidden units, and $\mathcal{L}_{L1}$ is the regularization loss in addition to the training loss.

A possible explanation for this effect is that the L1 regularizer has a similar effect as maximum likelihood estimation of an independent Laplace distribution. To see that, let

$$p_y(\mathbf{y}) \propto \exp\left(-\|\mathbf{y}\|_1\right)$$

and $\mathbf{x} = A\mathbf{y}$ with full rank invertible matrix A. Under this model

$$p_x(\mathbf{x}) = p_y(\mathbf{y}(\mathbf{x}))\,|\det A|^{-1}.$$

Then, minimization of the L1 norm of the activations under the constraint $\det A = \text{const.}$ corresponds to maximum likelihood on that model, which would encourage decorrelated responses. In our case, we do not enforce such a constraint, and the matrix might even not be invertible. However, the supervised loss function of the network benefits from having diverse non-zero filters. This encourages the network to not collapse filters on the same direction or put them to zero, and might act as a relaxation of the determinant constraint.

We evaluate the normalization schemes on three different tasks:

- **CNN image classification:** We apply different normalizations on CNNs trained on the CIFAR-10/100 datasets for image recognition, which each contains 50,000 training images and 10,000 test images. Each image is of size 32 × 32 × 3 and has been labeled an object class out of 10 or 100 total number of classes, respectively.

- **RNN language modeling:** We apply different normalizations on RNNs trained on the Penn Treebank dataset for language modeling, containing 42,068 training sentences, 3,370 validation sentences, and 3,761 test sentences.

- **CNN image super-resolution:** We train a CNN on low resolution images and learn cascades of nonlinear filters to smooth the upsampled images. We report performance of trained CNN on the standard Set 14 and Berkeley 200 dataset.

Table 10 and 11 summarizes the test performances of DN and DN* on CNNs and RNNs. CNNs are evaluated with test classification accuracy and RNNs are evaluated with test perplexity, which is defined as $\text{ppl} = \exp(-\sum_x \log p(x))$. We

Table 9. Different choices of the summation and suppression fields $\mathcal{A}$ and $\mathcal{B}$, as well as the constant σ in the normalizer lead to known normalization schemes in neural networks. $d(i, j)$ denotes an arbitrary distance between two hidden units i and j, and R denotes the neighbourhood radius.

Model	Range		Normalizer Bias
BN	$\mathcal{A}_{n,j} = \{z_{m,j} : m \in [1, N], j \in [1, H] \times [1, W]\}$	$\mathcal{B}_{n,j} = \{v_{m,j} : m \in [1, N], j \in [1, H] \times [1, W]\}$	$\sigma = 0$
LN	$\mathcal{A}_{n,j} = \{z_{n,i} : i \in [1, L]\}$	$\mathcal{B}_{n,j} = \{v_{n,i} : i \in [1, L]\}$	$\sigma = 0$
DN	$\mathcal{A}_{n,j} = \{z_{n,i} : d(i, j) \leq R_{\mathcal{A}}\}$	$\mathcal{B}_{n,j} = \{v_{n,i} : d(i, j) \leq R_{\mathcal{B}}\}$	$\sigma \geq 0$

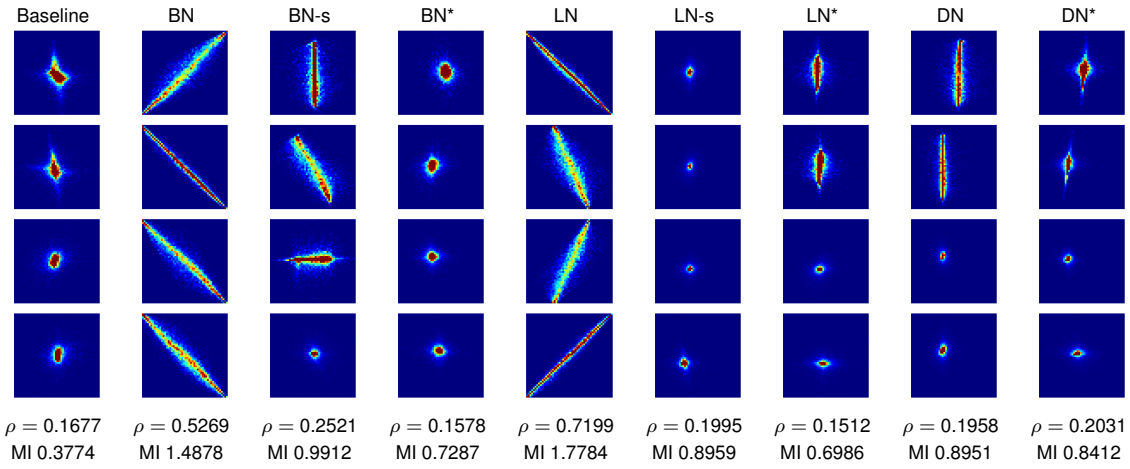


Fig. 11. First layer CNN pre-normalized activation joint histogram. All "-s" methods have an additional constant in the normalizer. DN always uses the constant in the normalizer. All "*" methods additionally use L1 regularization on the activities. In general, the L1 regularizer seems to encourage decorrelated activities. The bottom row shows the average pairwise mutual information (MI) and correlations (ρ) between all filters on that layer.

can see that DN and DN* outperforms a few baseline models, with dropout [76] and L2 regularization on the weights, and the standard batch and layer normalizations. L1 regularization (*) consistently improves the classification performance, especially for the original LN. The modification of LN makes it now better than the original BN, and only slightly worse than BN*. DN* achieves comparable performance to BN* on both datasets, but only relying on a local neighbourhood of hidden units, which is a significant advantage for online learning.

Table 10. Accuracy on CIFAR-10/100 experiments

Model	CIFAR-10 Acc.	CIFAR-100 Acc.
Baseline	0.7565	0.4409
Baseline + L2 + Dropout	0.7795	0.4179
BN	0.7807	0.4814
LN	0.7211	0.4249
DN	0.8058	0.4892
DN*	**0.8122**	**0.5066**

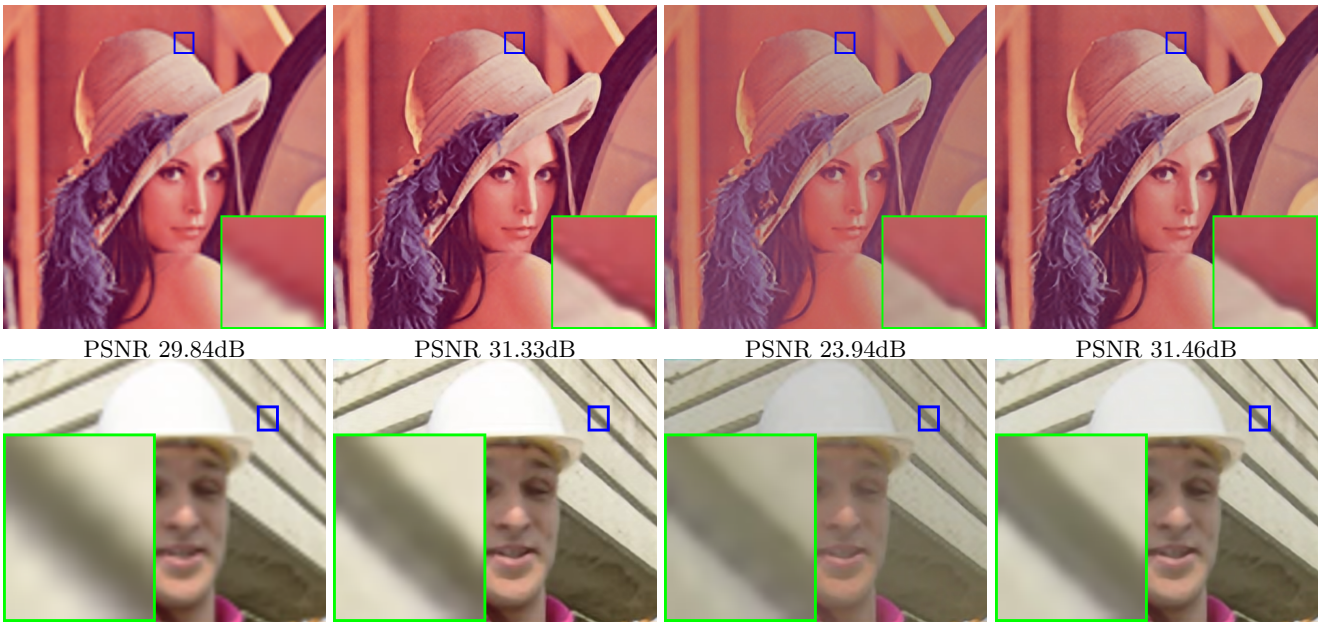
For the RNNs, we find that BN and LN alone do not improve the final performance relative to the baseline, but similar to what we see in the CNN experiments, our modified versions BN* and LN* show significant improvements. BN* on RNN is outperformed by both LN* and DN. By applying our normalization, we can improve the vanilla RNN perplexity by 20%, comparable to an LSTM baseline with the same hidden dimension.

Table 11. Perplexity measure on PTB word-level language modeling experiments

Model	LSTM	TanH RNN	ReLU RNN
Baseline	115.720	149.357	147.630
BN	123.245	148.052	164.977
LN	119.247	154.324	149.128
DN	103.714	132.143	118.789
DN*	**102.238**	**123.652**	**117.868**

For the super resolution experiments, we report the average test results, utilizing the standard metrics PSNR and SSIM [77], on two standard test datasets Set14 [78] and BSD200 [79]. We compare with two state-of-the-art single image super-resolution methods, A+ [80] and SRCNN [81]. All measures are computed on the Y channel of YCbCr color space. We also provide a visual comparison in Fig. 12.

As show in Tables 12 and 13, DN* outperforms the strong competitor SRCNN, while BN does not perform well on this task. The reason may be that BN applies the same statistics to all patches of one image which causes some overall intensity shift (see Figs. 12). From the visual comparisons, we can see that our method not only enhances the resolution but also removes artifacts like the ringing effect in Fig. 12.



(a) Bicubic (b) SRCNN (c) BN (d) DN*

Fig. 12. Super-resolution quality at magnification factor of 4. In super-resolution, a high resolution image is predicted from a downsampled version. We use peak signal-to-noise ratio to measure the algorithms' performances. Divisive normalization with sparsity regularization (DN*) outperforms the state-of-the-art convolutional network (SRCNN), while conventional batch normalization (BN) does not perform well on this task. From the visual comparisons, we can see that our method not only enhances the resolution but also removes artifacts, e.g. ringing effects.

Table 12. Average test results of PSNR and SSIM on Set14 Dataset.

Model	PSNR (x3)	SSIM (x3)	PSNR (x4)	SSIM (x4)
Bicubic	27.54	0.7733	26.01	0.7018
A+	29.13	0.8188	27.32	0.7491
SRCNN	**29.35**	0.8212	27.53	0.7512
BN	22.31	0.7530	21.40	0.6851
DN*	29.34	**0.8219**	**27.64**	**0.7562**

Table 13. Average test results of PSNR and SSIM on BSD200 Dataset.

Model	PSNR (x3)	SSIM (x3)	PSNR (x4)	SSIM (x4)
Bicubic	27.19	0.7636	25.92	0.6952
A+	27.05	0.7945	25.51	0.7171
SRCNN	28.42	0.8100	26.87	0.7378
BN	21.89	0.7553	21.53	0.6741
DN*	**28.44**	**0.8110**	**26.96**	**0.7428**

Divisive normalization as maximum a posteriori (MAP) estimate.

Let $\mathbf{x} = \mathbf{u} \cdot \sqrt{z}$ where $\mathbf{u}$ is isotropic Gaussian and z is inverse γ-distributed

$$p(z) \quad = \quad \frac{a^{\beta}}{2^{\beta}\Gamma(\beta)} z^{-\beta-1} \exp\left(-\frac{\alpha}{2z}\right).$$

Then $\mathbf{x}$ is multivariate t-distributed [55]. If we want to compute the MAP estimate for $\mathbf{u}$ given $\mathbf{x}$ we need to maximize

$$p(\mathbf{u}|\mathbf{x}) \quad \propto \quad p(\mathbf{x}|\mathbf{u})\, p(\mathbf{u}).$$

We provide a solution along the lines of [54] with the difference that we compute the MAP for $\mathbf{u}$ instead of z. The problem is, that $p(\mathbf{u}|\mathbf{x})$ is degenerate since $\mathbf{u}$ must be linearly dependent on $\mathbf{x}$. Therefore, to compute the MAP, we can assume that $\mathbf{x}$ is aligned with the first coordinate axis, which turns the problem into a scalar problem.

$$\begin{aligned}
&\operatorname{argmax}_u \log p(u|x) \\
&= \operatorname{argmax}_u \log p(x|u) + \log p(u) \\
&= \operatorname{argmax}_u \log p_z\left(\frac{x^2}{u^2}\right) + \underbrace{\log \frac{2x}{u^2}}_{\log \frac{\mathrm{d}z}{\mathrm{d}x}} + \log p(u) \\
&= \operatorname{argmax}_u -(\beta+1)\log\left(\frac{x^2}{u^2}\right) - \frac{\alpha u^2}{2x^2} + \log\frac{2x}{u^2} - \frac{u^2}{2\sigma^2}.
\end{aligned}$$

Taking the derivative yields

$$\frac{\mathrm{d}\log p(u|x)}{\mathrm{d}u} \quad \propto \quad \frac{2(\beta+1)-2}{u} - \frac{\alpha u}{x^2} - \frac{u}{\sigma^2}.$$

Setting it to zero and solving for u yields

$$u \quad = \quad \pm\frac{\sqrt{2\beta}\sigma x}{\sqrt{\alpha\sigma^2 + x^2}}.$$

Since we changed the axis such that $\mathbf{x}$ was on the first coordinate axis, we have $x = \|\mathbf{x}\|$ and we obtain divisive normalization as a MAP of the underlying Gaussian source.

Clustering.

In this section, we introduce our clustering based regularization which not only encourages the neural network to learn more compact representations, but also enables interpretability of the neural network.

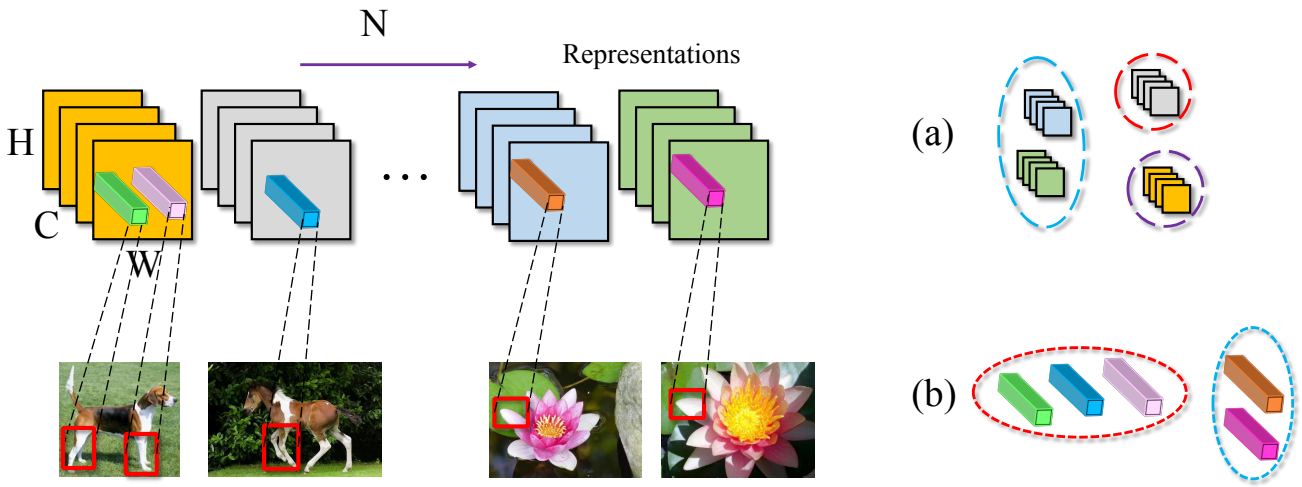


Fig. 13. (a) Sample clustering and (b) spatial clustering. Samples, pixels and channels are visualized as multi-channel maps, cubes and maps respectively. The receptive fields in the input image are denoted as red boxes.

We start our discussion by introducing some notation. We refer to $[K]$ as the set of K positive integers, *i.e.*, $[K] = \{1, 2, ..., K\}$. We use $\mathcal{S}\backslash\mathcal{A}$ to denote the set $\mathcal{S}$ with elements from the set $\mathcal{A}$ removed. A tensor is a multilinear map over a set of vector spaces. In tensor terminology, *n-mode* vectors of a D-order tensor $\mathbf{Y} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_D}$ are I_n-dimensional vectors obtained from $\mathbf{Y}$ by varying the index in I_n-dimension, while keeping all other indices fixed. An *n-mode* matrix unfolding of a tensor is a matrix which has all *n-mode* vectors as its columns [82]. Formally we use the operator $T^{\{I_n\}\times\{I_j|j\in[D]\backslash n\}}$ to denote the *n-mode* matrix unfolding, which returns a matrix of size $I_n \times \prod_{j\in[D]\backslash n} I_j$. Similarly, we define $T^{\{I_i,I_j\}\times\{I_k|k\in[D]\backslash\{i,j\}\}}$ to be an (i, j)-*mode* matrix unfolding operator. In this case a column vector is a concatenation of one *i-mode* vector and one *j-mode* vector. We denote the m-th row vector of a matrix $\mathbf{X}$ as $\mathbf{X}_m$. The representation of one layer within a neural network is denoted as a 4-D tensor $\mathbf{Y} \in \mathbb{R}^{N\times C\times H\times W}$, where N, C, H and W are the number of samples within a mini-batch, the number of hidden units, the height and width of the representation respectively. In the case of a fully connected layer, the dimensions along height and width become a singleton and the tensor degenerates to a matrix. Let $\mathcal{L}$ be the loss function of a neural network. In addition, we refer to the clustering regularization of a single layer via $\mathcal{R}$. The final objective is $\mathcal{L} + \lambda\mathcal{R}$, where λ adjusts the importance of the clustering regularization. We can add a regularization term for any subset of layers, but we focus on a single layer for notational simplicity. In what follows, we show three different types of clustering, each possessing different properties. In our framework any variant can be applied to any layer.

Sample Clustering: We first investigate clustering along the sample dimension. We refer the reader to Fig. 13 (a) for an illustration. In particular, given the representation tensor $\mathbf{Y}$, we first unfold it into a matrix $T^{\{N\}\times\{H,W,C\}}(\mathbf{Y}) \in \mathbb{R}^{N\times HWC}$. We then encourage the samples to cluster as follows:

$$\mathcal{R}_{sample}(\mathbf{Y}, \mu) = \frac{1}{2NCHW} \sum_{n=1}^{N} \left\| T^{\{N\}\times\{H,W,C\}}(\mathbf{Y})_n - \mu_{z_n} \right\|^2, \quad [14]$$

where μ is a matrix of size $K \times HWC$ encoding all cluster centers, with K the total number of clusters. $z_n \in [K]$ is a discrete latent variable corresponding to the n-th sample. It indicates which cluster this sample belongs to. Note that for a fully connected layer, the formulation is the same except that $T^{\{N\}\times\{H,W,C\}}(\mathbf{Y})_n$ and μ_{z_n} are C-sized vectors since $H = W = 1$ in this case.

Spatial Clustering: The representation of one sample can be regarded as a C-channel "image." Each spatial location within that "image" can be thought of as a "pixel," and is a vector of size C (shown as a colored bar in Fig. 13). For a ConvNet, every "pixel" has a corresponding receptive field covering a local region in the input image. Therefore, by clustering "pixels" of all images during learning, we expect to model local parts shared by multiple objects or scenes. To achieve this, we adopt the unfolding operator $T^{\{N,H,W\}\times\{C\}}(\mathbf{Y})$ and use

$$\mathcal{R}_{spatial}(\mathbf{Y}, \mu) = \frac{1}{2NCHW} \sum_{i=1}^{NHW} \| T^{\{N,H,W\}\times\{C\}}(\mathbf{Y})_i - \mu_{z_i} \|^2. \quad [15]$$

Channel Co-Clustering: This regularizer groups the channels of different samples directly, thus co-clustering samples and filters. We expect this type of regularization to model reoccurring patterns shared not only among different samples but also within each sample. Relying on the unfolding operator $T^{\{N,C\}\times\{H,W\}}(\mathbf{Y})$, we formulate this type of clustering objective as

$$\mathcal{R}_{channel}(\mathbf{Y}, \mu) = \frac{1}{2NCHW} \sum_{i=1}^{NC} \| T^{\{N,C\}\times\{H,W\}}(\mathbf{Y})_i - \mu_{z_i} \|^2. \quad [16]$$

Performance We use CIFAR10 and CIFAR100 datasets [83] to benchmark our clustering regularizer. CIFAR10 consists of 60,000 32×32 images assigned to 10 categories. CIFAR100 in contrast differentiates between 100 classes. We use the standard split on both datasets. The quick CIFAR10 architecture of Caffe [84] is used for benchmarking both datasets. It consists of 3 convolutional layers and 1 fully connected layer followed by a softmax layer. We report mean accuracy

averaged over 4 trials. For fully connected layers we use the sample-clustering objective as the representation is a matrix. For convolutional layers, we provide the results of all three different clustering objectives. We refer to the three approaches as 'sample-clustering,' 'spatial-clustering,' and 'channel-co-clustering' respectively. We set all hyper parameters based on cross-validation.

In Table 14 we compare our framework with some recent regularizers, like DeCov [85], Dropout [86] and the baseline results obtained by using Caffe. We again observe that all of our methods achieve better generalization performance.

Siamese networks based on VGG.

We have experimented with constructing a Siamese network on top of the commonly used VGG-16 architecture. For this, we extract features from the topmost convolutional layer of VGG-16 by average pooling these 7x7x512 representations to produce 512-dimensional embedding vectors. We combine the representations from the two branches by taking the element-wise absolute difference, yielding another 512-dimensional vector that is subsequently fed into a binary classifier that predicts whether or not the two input images are similar. We backpropagate the error of the cross entropy loss of this classifier through the weights of the VGG architecture to learn our embedding function.

In mathematical terms, let $f(x)$: image $\rightarrow \mathbb{R}^F$ be the embedding function employed by each of the two branches of the network, parameterized by the convolutional layers of a VGG-16, which takes an image x and returns a feature vector of dimensionality F. Let x_1^i and x_2^i denote the first and second image of the i_{th} training pair, respectively. Then the prediction p^i is given by

$$p^i = \text{softmax}(|f(x_1^i) - f(x_2^i)|) \qquad [17]$$

Then, given P training pairs, and $\forall i \in 1, \ldots, |P|$ the pair prediction p^i and the ground-truth similarity label y_i, the cross-entropy loss function is given below.

$$L = -\sum_{i=1}^{P} y_i \log p_i + (1 - y_i) \log(1 - p_i) \qquad [18]$$

We construct our training batches by first randomly sampling N classes uniformly at random that will participate in the batch and then sampling batch size many images uniformly at random from the pool of images belonging in the N chosen classes. Training the Siamese network then proceeds by forming all possible pairs from within these sampled images. The hyperparameters that this method introduces are the number of batch images and the number of batch classes. Our best-performing variant was trained with 32 images in each batch, coming from 4 different classes.

Training Details for the DRMM-Siamese networks.

The details of the DRMM trainings and the values of hyper-parameters are provided in Table 15. The models are trained using RMSProp [87] with exponentially-decayed learning rate. The learning rate starts at 0.001 and ends at 0.000001 over 500 epochs. The decay rate is applied on the learning rate after each epoch. We initialize the parameter dictionary $\Lambda^{(\ell)}_{g^{(\ell)}}$ at each layer ℓ using Xavier initialization. The implementation of the DRMM generation process can be found in Section 7.

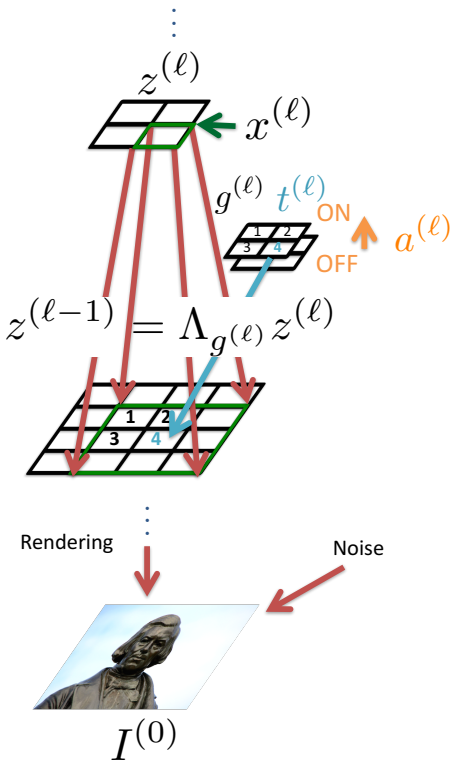


Fig. 14. DRMM generative model: When $a^{(\ell)}_{x^{(\ell)}} = 1$ (ON) (light orange), a single super pixel $x^{(\ell)}$ at level ℓ (green, upper) renders down to a 3×3 image patch at level $\ell - 1$ (green, lower), whose location is specified by $t^{(\ell)}_{x^{(\ell)}}$ (light blue). If $a^{(\ell)}_{x^{(\ell)}} = 0$ (OFF), there is no rendering at $x^{(\ell)}$. $I^{(0)} = I$ is the final rendered image. Note that only the transformation from level ℓ of the hierarchy of abstraction to level $\ell - 1$ is shown here.

For the object recognition task, the set of labeled images is replicated until its size is the same as the size of the unlabeled set. In each training iteration, the same amounts of the labeled and unlabeled images (half of the batch size) are sent into the DRMM. The batch size used is 200. Note that we only apply divisive normalization after the convolutions, but not after the pooling layers. Also, we do not apply divisive normalization in the Top-Down reconstruction. For the similarity finetuning, we first train our DRMM-Siamese network to minimize cross-entropy loss until convergence, and then we keep training the network to minimize the linear combination of the cross-entropy and contrastive loss with the loss weights $\alpha_{cross-entropy} = 1.0$ and $\alpha_{contrastive} = 0.1$.

Generation Process in the Deep Rendering Mixture Model.

The generation in the DRMM takes the form:

$$\begin{aligned} c^{(L)} &\sim \text{Cat}(\{\pi_{c^{(L)}}\}) \\ g^{(\ell)} &\sim \text{Cat}(\{\pi_{g^{(\ell)}}\}) \\ \mu_{cg} &\equiv \Lambda_g \mu_{c^{(L)}} \\ &\equiv \Lambda^{(1)}_{g^{(1)}} \Lambda^{(2)}_{g^{(2)}} \ldots \Lambda^{(L-1)}_{g^{(L-1)}} \Lambda^{(L)}_{g^{(L)}} \mu_{c^{(L)}} \\ I &\sim \mathcal{N}(\mu_{cg}, \sigma^2 \mathbf{1}_{D^{(0)}}), \end{aligned}$$

where $\ell \in \{1, 2, \ldots, L\}$ is the layer, $c^{(L)}$ is the object category, $g^{(\ell)}$ are the latent (nuisance) variables at layer ℓ, and

Table 14. CIFAR10 and CIFAR 100 results. For DeCov, no standard deviation is provided for the CIFAR100 results [85]. All our approaches outperform the baselines.

Dataset	CIFAR10 Train	CIFAR10 Test	CIFAR100 Train	CIFAR100 Test
Caffe	94.87 ± 0.14	76.32 ± 0.17	68.01 ± 0.64	46.21 ± 0.34
Weight Decay	95.34 ± 0.27	76.79 ± 0.31	69.32 ± 0.51	46.93 ± 0.42
DeCov	88.78 ± 0.23	79.72 ± 0.14	77.92	40.34
Dropout	99.10 ± 0.17	77.45 ± 0.21	60.77 ± 0.47	48.70 ± 0.38
Sample-Clustering	89.93 ± 0.19	**81.05** ± 0.41	63.60 ± 0.55	**50.50** ± 0.38
Spatial-Clustering	90.50 ± 0.05	**81.02** ± 0.12	64.38 ± 0.38	**50.18** ± 0.49
Channel Co-Clustering	89.26 ± 0.25	**80.65** ± 0.23	63.42 ± 1.34	**49.80** ± 0.25

Task	Optimiser/Hyper-parameters
Tiny ImageNet Object recognition pretraining	RMSProp learning rate 0.001 - 0.000001 over 500 epochs $\alpha_H = 1.0, \alpha_{RC} = 0.2, \alpha_{KL} = 1.0, \alpha_{NN} = 1.0$ batch size = 200
Benchmark Object recognition pretraining	RMSProp learning rate 0.001 - 0.000001 over 500 epochs $\alpha_H = 1.0, \alpha_{RC} = 0.2, \alpha_{KL} = 0.5, \alpha_{NN} = 0.5$ batch size = 200
Benchmark Similarity finetuning	RMSProp learning rate 0.001 - 0.000001 over 500 epochs $\alpha_{cross-entropy} = 1.0, \alpha_{contrastive} = 0.1$ batch size = 200

Table 15. Details of the DRMM trainings and values of the hyperparameters.

$\Lambda^{(\ell)}_{g^{(\ell)}} \in \mathbb{R}^{D^{(\ell-1)} \times D^{(\ell)}}$ are parameter dictionaries that contain templates at layer ℓ. The image I is generated by adding isotropic Gaussian noise to a multiscale "rendered" template μ_{cg}, and $D^{(0)}$ is the dimension of the image I. When applying the hard Expectation-Maximization (EM) algorithm, we take the zero-noise limit. Here, $g^{(\ell)} = \left(t^{(\ell)}, a^{(\ell)}\right)$ where $a^{(\ell)} \equiv \left(a^{(\ell)}_{x^{(\ell)}}\right)_{x^{(\ell)} \in \mathcal{X}^{(\ell)}}$ is a vector of binary switching variables that select the templates to render and $t^{(\ell)} \equiv \left(t^{(\ell)}_{x^{(\ell)}}\right)_{x^{(\ell)} \in \mathcal{X}^{(\ell)}}$ is the vector of rendering positions. Both $a^{(\ell)}$ and $t^{(\ell)}$ have dimension $D^{(\ell)}$. Note that $x^{(\ell)} \in \mathcal{X}^{(\ell)} \equiv \{\text{pixels in level } \ell\}$ (see Figure 14) and $t^{(\ell)}_{x^{(\ell)}} \in \{\text{UL}, \text{UR}, \text{LL}, \text{LR}\}$ where UL, UR, LL and LR stand for upper left, upper right, lower left and lower right positions, respectively. As defined in [12] (see also Figure 14), the intermediate rendered image $z^{(\ell-1)}$ is given by:

$$z^{(\ell-1)} \equiv \Lambda^{(\ell)}_{g^{(\ell)}} z^{(\ell)} \quad [19]$$

$$= \Lambda^{(\ell)}_{t^{(\ell)}, a^{(\ell)}} z^{(\ell)} \quad [20]$$

$$= \sum_{x^{(\ell)} \in \mathcal{X}^{(\ell)}} T^{(\ell)}_{t^{(\ell)}_{x^{(\ell)}}} Z^{(\ell)}_{x^{(\ell)}} \left(\Gamma^{(\ell)} M^{(\ell)}_{a^{(\ell)}}\right)_{x^{(\ell)}} z^{(\ell)}_{x^{(\ell)}} \quad [21]$$

$$= \text{DRMMLayer}(z^{(\ell)}, t^{(\ell)}, a^{(\ell)}, \Gamma^{(\ell)}) \quad [22]$$

where $M^{(\ell)}_{a^{(\ell)}} \equiv \text{diag}\left(a^{(\ell)}\right) \in \mathbb{R}^{D^{(\ell)} \times D^{(\ell)}}$ is a masking matrix, $\Gamma^{(\ell)} \in \mathbb{R}^{F^{(\ell)} \times D^{(\ell)}}$ is the set of core templates of size $F^{(\ell)}$ (without any zero-padding and translation) at layer ℓ, $Z^{(\ell)}$ is a set of zero-padding operators, and $T^{(\ell)}_{t^{(\ell)}}$ is a set of translation operators to position $t^{(\ell)}$. Elements of $Z^{(\ell)}$ and $T^{(\ell)}_{t^{(\ell)}}$ are indexed by $x^{(\ell)}$, $Z^{(\ell)}_{x^{(\ell)}} \in \mathbb{R}^{D^{(\ell-1)} \times F^{(\ell)}}$ and $T^{(\ell)}_{t^{(\ell)}_{x^{(\ell)}}} \in \mathbb{R}^{D^{(\ell-1)} \times D^{(\ell-1)}}$. Also, $\Gamma^{(\ell)}[:, x^{(\ell)}]$ are the same for $x^{(\ell)}$ in the same channel of the intermediate rendered image $z^{(\ell)}$. Note that in the main paper, we call $z^{(\ell-1)}$ and $z^{(\ell)}$ intermediate rendered templates.

The DRMM layer can be implemented using convolutions of filters $\Gamma^{(\ell)}$, or equivalently, deconvolutions of filters $\Gamma^{(\ell)T}$. $a^{(\ell)}$ and $t^{(\ell)}$ are used to select rendering templates and positions to render, respectively. In the E-step Top-Down Reconstruction, $\hat{t}^{(\ell)}$ and $\hat{a}^{(\ell)}$ estimated in the E-step Bottom-Up are used instead.

More Pruning Results.

Supervised Learning. We investigate the LeNet-5 under synaptic and neuron pruning by studying the number of parameters and floating-point (FLOP computations in each layer of the network and compare with the similar analysis for Deep Compression pruning [59]. The results are shown in Table 16, 17 and 18. For each layer of the LeNet-5, those tables show the original number of weights before pruning, the original number of floating-point operations required to compute that layer's activations, the average percentage of the activations (after max-pooling) that are non-zero, the percentage of non-zero weights after pruning, and the percentage of the floating-point operations actually required. For the neuron pruning, we also show the percentage of the switching variables a that are still ON after the pruning. Both synapse and neuron pruning are slightly better than the Deep Compression pruning in the FLOP reduction rate.

Table 16. For Lenet-5, pruning technique in Deep Compression reduces the number of weights by $12\times$ and computation by $6\times$

Layer	Weights	FLOP	Act%	Weights%	FLOP%
conv1	0.5K	576K	82%	66%	66%
conv2	25K	3200K	72%	12%	10%
fc1	400K	800K	55%	8%	6%
fc2	5K	10K	100%	19%	10%
Total	431K	4586K	77%	**8%**	**16%**

Table 17. For Lenet-5, our synapse pruning reduces the number of weights by $12\times$ and computation by $8\times$

Layer	Weights	FLOP	Act%	Weights%	FLOP%
conv1	0.5K	576K	88%	66%	66%
conv2	25K	3200K	38%	7%	6%
fc1	400K	800K	53%	8%	3%
fc2	5K	10K	100%	26%	14%
Total	431K	4586K	74%	**8%**	**13%**

Table 18. For Lenet-5, our neuron pruning reduces the number of computation by $8\times$

Layer	FLOP	Act%	Permanent Active a%	FLOP%
conv1	576K	55%	59%	59%
conv2	3200K	13%	10%	6%
fc1	800K	14%	19%	2%
fc2	10K	100%	100%	14%
Total	4586K	42%	47%	**12%**

Semi-Supervised Learning. In addition to the statistics included Table 16, 17 and 18, we report the FLOP operations required for the Bottom-Up (BU) Inference the Top-Down (TD) Reconstruction seperately in Table 19, 20, and 21. We also report the average percentage of pixels in the intermediate rendered images Z that are non-zero (Active Top-Down Pixels %).

Table 19. For the 5-layer DRMM, pruning technique in Deep Compression reduces the number of weights by $3\times$ and computation by $8.3\times$

Layer	Weights	FLOP BU	FLOP TD	Act%	Active Top-Down Pixels%	Weights%	FLOP%
conv1	0.8K	1,638K	1,254K	40%	10%	36%	22%
conv2	18.43K	7,225K	9,437K	41%	41%	33%	13%
conv3	36.86K	18,874K	14,451K	63%	16%	35%	11%
conv4	73.73K	5,308K	9,437K	35%	35%	25%	11%
conv5	1.28K	92K	92K	100%	56%	100%	46%
softmax	0.1K	0.2K	0.02K	100%	100%	100%	100%
Total	131.2K	33,137K	34,671K	43%	19%	**30%**	**12%**

Table 20. For the 5-layer DRMM, our synapse pruning reduces the number of weights by $1.7\times$ and computation by $8.3\times$

Layer	Weights	FLOP BU	FLOP TD	Act%	Active Top-Down Pixels%	Weights%	FLOP%
conv1	0.8K	1,638K	1,254K	15%	4%	25%	15%
conv2	18.43K	7,225K	9,437K	30%	25%	35%	7%
conv3	36.86K	18,874K	14,451K	56%	14%	20%	5%
conv4	73.73K	5,308K	9,437K	28%	27%	84%	31%
conv5	1.28K	92K	92K	100%	57%	100%	43%
softmax	0.1K	0.2K	0.02K	100%	100%	100%	100%
Total	131.2K	33,137K	34,671K	29%	12%	**59%**	**12%**

Table 21. For the 5-layer DRMM, our neuron pruning reduces the number of computation by $8.3\times$

Layer	FLOP BU	FLOP TD	Act%	Active Top-Down Pixels%	Permanent Active a%	FLOP%
conv1	1,638K	1,254K	30%	7%	33%	9%
conv2	7,225K	9,437K	26%	26%	38%	19%
conv3	18,874K	14,451K	45%	11%	38%	14%
conv4	5,308K	9,437K	1%	1%	4%	0.6%
conv5	92K	92K	100%	33%	100%	67%
softmax	0.2K	0.02K	100%	100%	100%	100%
Total	33,137K	34,671K	27%	11%	**34%**	**12%**