Towards Biologically Plausible Convolutional Networks

Anonymous Author(s)
Affiliation
Address
email

Abstract

Convolutional networks are ubiquitous in deep learning. They are particularly useful for images, as they reduce the number of parameters, reduce training time, and increase accuracy. However, as a model of the brain they are seriously problematic, since they require weight sharing – something real neurons simply cannot do. Consequently, while neurons in the brain can be locally connected (one of the features of convolutional networks), they cannot be convolutional. Locally connected but non-convolutional networks, however, significantly underperform convolutional ones. This is troublesome for studies that use convolutional networks to explain activity in the visual system. Here we study plausible alternatives to weight sharing that aim at the same regularization principle, which is to make each neuron within a pool react similarly to identical inputs. The most natural way to do that is by showing the network multiple translations of the same image, akin to saccades in animal vision. However, this approach requires many translations, and doesn't remove the performance gap. We propose instead to add lateral connectivity to a locally connected network, and allow learning via Hebbian plasticity. This requires the network to pause occasionally for a sleep-like phase of "weight sharing". This method enables locally connected networks to achieve nearly convolutional performance on ImageNet, thus supporting convolutional networks as a model of the visual stream.

1 Introduction

Convolutional networks are a cornerstone of modern deep learning: they're widely used in the visual domain [1, 2, 3], speech recognition [4], text classification [5], and time series classification [6] – basically, in any system that is approximately translation invariant. They have also played an important role in enhancing our understanding of the visual stream [7]. Indeed, simple and complex cells in the visual cortex [8] inspired convolutional and pooling layers in deep networks [9] (with simple cells implemented with convolution and complex ones with pooling). Moreover, the representations found in convolutional networks are similar to those in the visual stream [10, 11, 12, 13] (see [7] for an in-depth review).

Despite the success of convolutional networks at reproducing activity in the visual system, as a model of the visual system they are somewhat problematic. That's because convolutional networks share weights, something biological networks, for which weight updates must be local, can't do [14]. Locally connected networks avoid this problem by using the same receptive fields as convolutional networks (thus locally connected), but without weight sharing [15]. However, they pay a price for biological plausibility: locally connected networks are known to perform worse than their convolutional counterparts on hard image classification tasks [15, 16]. There is, therefore, a need for a mechanism to bridge the gap between biologically plausible locally connected networks and implausible convolutional ones.

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

Here, we consider two such mechanisms. One is to use extensive data augmentation (primarily image translations); the other is to introduce an auxiliary objective that allows some form of weight sharing, which is implemented by lateral connections; we call this approach dynamic weight sharing.

The first approach, data augmentation, is simple, but we show that it suffers from two problems: it requires far more training data than is normally used, and even then it fails to close the performance gap between convolutional and locally connected networks. The second approach, dynamic weight sharing, implements a sleep-like phase in which neural dynamics facilitate weight sharing. This is done through lateral connections in each layer, which allows subgroups of neurons to share their activity. Through this lateral connectivity, each layer can first equalize its weights via anti-Hebbian learning, and then generate an input pattern for the next layer that helps it to do the same thing. Dynamic weight sharing doesn't achieve perfectly convolutional connectivity, as only subgroups of neurons share weights. However, it implements a similar inductive bias, and, as we show in experiments, it performs almost as well as convolutional networks.

Our study suggests that convolutional networks may be biologically plausible as they can be approximated in realistic networks simply by adding lateral connectivity and Hebbian learning. As the performance of convolutional networks and locally connected networks with dynamic weight sharing is similar, convolutional networks remain a good "model organism". This is important, as they consume much less memory than locally connected networks, and run much faster on modern hardware.

2 Related work

Studying systems neuroscience through the lens of deep learning is an active area of research, especially when it comes to the visual system [17]. As mentioned above, convolutional networks in particular have been extensively studied as a model of the visual stream (and also inspired by it) [7], and also as mentioned above, because they require weight sharing they lack biological plausibility. They also have been widely used to evaluate the performance of different biologically plausible learning rules [18, 15, 19, 20, 21, 22, 23, 24].

Several studies have tried to relax weight sharing in convolutions by introducing locally connected networks [15, 16, 25] ([25] also shows that local connectivity itself can be learned from a fully connected network with proper weight regularization). Locally connected networks perform as well as convolutional ones in shallow architectures [15, 25]. However, they perform worse for large networks and hard tasks, unless they're initialized from an already well-performing convolutional solution [16] or have some degree of weight sharing [26]. In this study, we seek biologically plausible regularizations of locally connected network to improve their performance.

Convolutional networks are not the only deep learning architecture for vision: visual transformers (e.g., [27, 28, 29, 30]), and more recently, the transformer-like architectures without self-attention [31, 32, 33], have shown competitive results. However, they still need weight sharing: at each block the input image is reshaped into patches, and then the same weight is used for all patches. Our Hebbian-based approach to weight sharing fits this computation as well.

3 Regularization in locally connected networks

3.1 Convolutional versus locally connected networks

Convolutional networks are implemented by letting the weights depend on the difference in indices. Considering, for simplicity, one dimensional convolutions and a linear network. Letting the input and output of a one layer in a network be x_j and z_i, respectively, the update rule in a convolutional network is

$$z_i = \sum_{j=1}^{N} w_{i-j} x_j \,, \tag{1}$$

where N is the number of neurons; for definiteness, we'll assume N is the same in in each layer (right panel in Fig. 1). Although the index j ranges over all N neurons, many, if not most, of the weights are zero: w_{i-j} is nonzero only when $|i-j| \leq k/2 < N$ for the kernel size k.

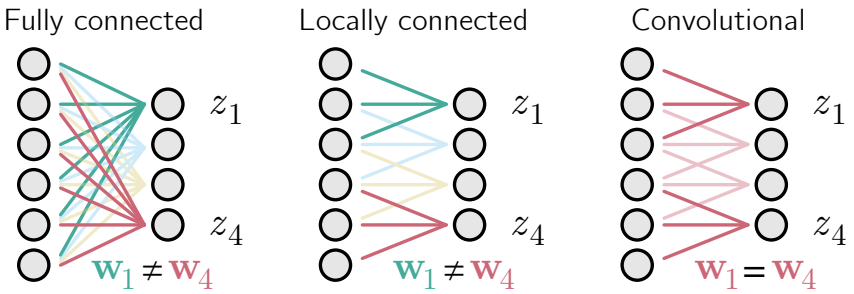

Figure 1: Comparison between layer architectures: fully connected (left), locally connected (middle) and convolutional (right). Locally connected layers do no share weights among neurons (like fully connected ones), but have the same receptive fields as convolutional layers.

For networks that aren't convolutional, the weight matrix w_{i-j} is replaced by w_{ij},

$$z_i = \sum_{j=1}^{N} w_{ij} x_j \,. \tag{2}$$

Again, the index j ranges over all N neurons. If all the weights are nonzero, the network is fully connected (left panel in Fig. 1). But, as in convolutional networks, we can restrict the connectivity range by letting w_{ij} be nonzero only when $|i-j| \leq k/2 < N$, resulting in a locally connected (but still non-convolutional) network (center panel in Fig. 1).

3.2 Developing convolutional weights: data augmentation versus dynamic weight sharing

Here we explore the question: is it possible for a locally connected network to develop approximately convolutional weights? That is, after training, is it possible to have $w_{ij} \approx w_{i-j}$? There is one straightforward way to do this: augment the data to provide multiple translations of the same image, so the network learns to react similarly to them (Fig. 2A). A potential problem is that a large number of translations will be needed. This makes training costly (see Section 5), and is unlikely to be consistent with animal learning, as animals see only a handful of translations of any one image.

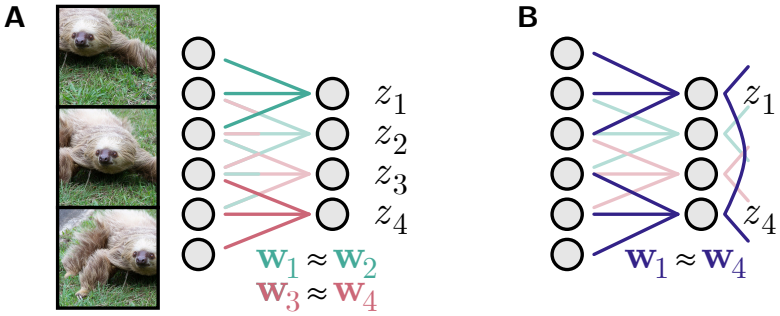

Figure 2: Two regularization strategies for locally connected networks. **A.** Data augmentation, where multiple translations of the same image are presented simultaneously. **B.** Dynamic weight sharing, where a subset of neurons equalizes their weights through lateral connections and learning.

A less obvious solution is to modify the network so that during learning the weights become approximately convolutional. As we show in the next section, this can be done by adding lateral connections, and introducing a "sleep" phase during training (Fig. 2B). This solution doesn't need more data, but it does need an additional training step.

4 A Hebbian solution to dynamic weight sharing

If we were to train a locally connected network without any weight sharing or data augmentation, the weights of different neurons would diverge (region marked "training" in Fig. 3A). Our strategy to

make them convolutional is to introduce an occasional "sleep" phase, during which the weights relax to their mean (region marked "sleep" in Fig. 3A). This will compensate weight divergence during learning by convergence during the sleep phase. If the latter is sufficiently strong, the weights will remain approximately convolutional.

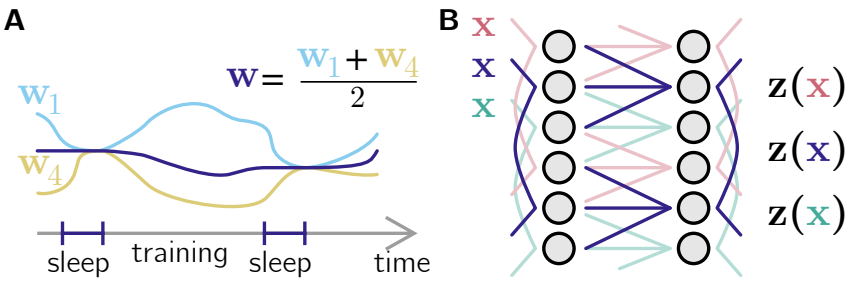

Figure 3: **A.** Dynamical weight sharing interrupts the main training loop, and equalizes the weights through internal dynamics. After that, the weights start diverging again until the next weight sharing phase. **B.** A locally connected network, where both input and output neurons have lateral connections. The input layer uses it to generate data, and the output layer – to share weights.

To implement this, we introduce lateral connectivity, chosen to equalize activity in both the input and output layer. That's shown in Fig. 3B, where every third neuron in both the input ($\mathbf{x}$) and output ($\mathbf{z}$) layers are connected. Once the connected neurons in the input layer have equal activity, all output neurons receive identical input. Since the lateral output connections also equalize activity, all connection that correspond to a translation by three neurons see exactly the same pre and postsynaptic activity. A naive Hebbian learning rule (with weight decay) would, therefore, make the network convolutional. However, we have to take care that during Hebbian learning, the initial weighs are not over-written. We now describe how that is done.

To ease notation, we'll let $\mathbf{w}_i$ be a vector containing the incoming weights to neuron i: $(\mathbf{w}_i)_j \equiv w_{ij}$. Moreover, we'll let j run from 1 to k, independent of i. With this convention, the response of neuron i, z_i, to a k-dimensional input, $\mathbf{x}$, is given by

$$z_i = \mathbf{w}_i \cdot \mathbf{x} = \sum_{j=1}^{k} w_{ij} x_j \,. \tag{3}$$

Consider the following update rule for the weights, where we assume that every neuron sees the same $\mathbf{x}$

$$\Delta \mathbf{w}_i \propto - \left(z_i - \frac{1}{N} \sum_{j=1}^{N} z_j \right) \mathbf{x} - \gamma \left(\mathbf{w}_i - \mathbf{w}_i^{\text{init}} \right) \tag{4}$$

where $\mathbf{w}_i^{\text{init}}$ are the weights at the beginning of the dynamic weight sharing cycle, and not before the overall training.

This Hebbian update effectively implements SGD over the sum of $(z_i - z_j)^2$, plus a regularizer (the second term) to keep the weights near $\mathbf{w}_i^{\text{init}}$. If we present the network with M different input vectors, $\mathbf{x}_m$, and denote the covariance matrix $\mathbf{C} \equiv \frac{1}{M} \sum_m \mathbf{x}_m \mathbf{x}_m^\top$, then the weight dynamics in Eq. (4) converges to (see Appendix A for the derivation)

$$\mathbf{w}_i^* = (\mathbf{C} + \gamma \, \mathbf{I})^{-1} \cdot \left(\mathbf{C} \cdot \frac{1}{N} \sum_{i=1}^{N} \mathbf{w}_i^{\text{init}} + \gamma \, \mathbf{w}_i^{\text{init}} \right) \,, \tag{5}$$

where $\mathbf{I}$ is the identity matrix.

This is exactly what we want, since as long as $\mathbf{C}$ is full rank and γ is small, we arrive at shared weights: $\mathbf{w}_i^* \approx \frac{1}{N} \sum_{i=1}^{N} \mathbf{w}_i^{\text{init}}$. It might seem advantageous to set $\gamma = 0$, as non-zero γ only biases the equilibrium value of the weight. However, non-zero γ ensures that for noisy input, $\mathbf{x}_i = \mathbf{x} + \xi_i$ (which is much more realistic), the weights still converge (at least approximately) to the mean of the initial weights (see Appendix A).

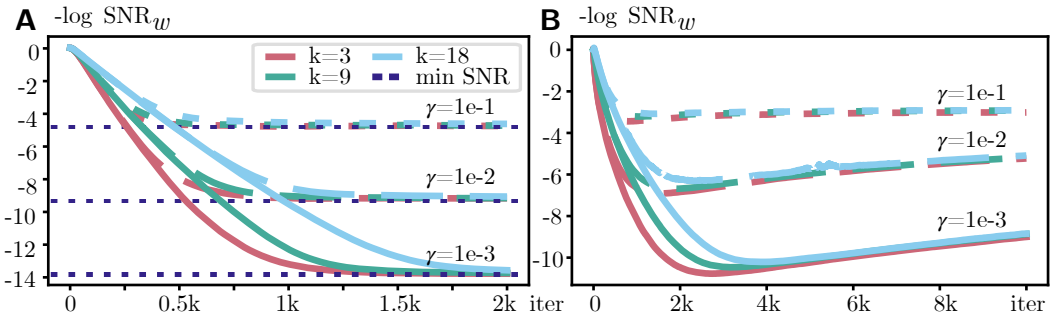

Figure 4: Logarithm of inverse signal-to-noise ratio (mean weight squared over weight variance, see Eq. (6)) for weight sharing objectives in a layer with 100 neurons. **A.** Dynamics of Eq. (4) for different kernel sizes k (meaning k^2 inputs) and γ. Dark dotted lines show the theoretical minimum. **B.** Dynamics of weight update that uses Eq. (9) for $\alpha = 10$, different kernel sizes k and γ. In each iteration, the input is presented for 150 ms.

In practice, the dynamics in Eq. (4) converges quickly. We illustrate it in Fig. 4A by plotting $-\log \mathrm{SNR}_w$ over time, where SNR_w, the signal to noise ratio of the weights, is defined as

$$\mathrm{SNR}_w = \frac{1}{k^2} \sum_j \frac{\left(\frac{1}{N}\sum_i (\mathbf{w}_i)_j\right)^2}{\frac{1}{N}\sum_i \left((\mathbf{w}_i)_j - \frac{1}{N}\sum_{i'} (\mathbf{w}_{i'})_j\right)^2} \,. \tag{6}$$

For all kernel sizes (we used 2d inputs, meaning k^2 inputs per neuron), the weight converge to a nearly convolutional solution within a few hundred iterations (note the logarithmic scale of the y axis in Fig. 4A). See Appendix A for simulation details. In order to run our experiments with deep networks in a realistic time frame, we perform weight sharing directly during the sleep phase.

4.1 Dynamic weight sharing in convolutional layers

As shown in Fig. 3B, the k-dimensional input, $\mathbf{x}$, repeats every k neurons. Consequently, during the sleep phase, the weights are not set to the mean of their initial value averaged across all neurons; instead, they're set to the mean averaged across a set of neurons spaced by k. Thus, in one dimension, the sleep phase equilibrates the weights in k different modules. In two dimensions (the realistic case), the sleep phase equilibrates the weights in k^2 different modules.

In multiple layers, the sleep phase is implemented layer by layer. In layer l, lateral connectivity creates repeated input patterns (e.g., by randomly activating some of the subgrid on the left of Fig. 3B) and feeds them to layer $l+1$. After weight sharing in layer $l+1$, the new pattern from $l+1$ is fed to $l+2$, and so on. We assume that the first layer can receive suitable patterns from the retina. However, this pattern generation scheme needs layers to have receptive fields of the same size.

4.2 A realistic model that implements the update rule

Our update rule, Eq. (4), implies a linear neuron which firing rate r_i that depends on the upstream input $z_i = w_i^\top \mathbf{x}$ and lateral input as

$$r_i = z_i - \frac{1}{N}\sum_{j=1}^{N} z_j \equiv \mathbf{w}_i^\top \mathbf{x} - \frac{1}{N}\sum_{j=1}^{N} \mathbf{w}_j^\top \mathbf{x} \,, \tag{7}$$

making an anti-Hebbian weight change $-r_i\,\mathbf{x}$. In a realistic circuit, this can be implemented with excitatory neurons r_i, and an inhibitory neuron r_{inh}, which obey the dynamics

$$\tau \dot{r}_i = -r_i + \mathbf{w}_i^\top \mathbf{x} - \alpha\, r_{\mathrm{inh}} + b \,, \tag{8a}$$

$$\tau \dot{r}_{\mathrm{inh}} = -r_{\mathrm{inh}} + \frac{1}{N}\sum_j r_j - b \,, \tag{8b}$$

where b is the shared bias term that ensures non-negativity of firing rates (assuming $\sum_i \mathbf{w}_i^\top \mathbf{x}$ is positive, which would be the case for excitatory input neurons). The only fixed point of this equations is

$$r_i^* = b + \mathbf{w}_i^\top \mathbf{x} - \frac{1}{N}\sum_j \mathbf{w}_j^\top \mathbf{x} + \frac{1}{1+\alpha}\frac{1}{N}\sum_j \mathbf{w}_j^\top \mathbf{x} \underset{\alpha \gg 1}{\approx} b + \mathbf{w}_i^\top \mathbf{x} - \frac{1}{N}\sum_j \mathbf{w}_j^\top \mathbf{x}\,, \tag{9}$$

which is stable. As a result, for strong inhibition ($\alpha \gg 1$), Eq. (4) can be implemented with an anti-Hebbian term $-(r_i - b)\mathbf{x}$. Note that if $\mathbf{w}_i \cdot \mathbf{x}$ is zero on average, then b is the mean firing rate over time. To show that Eq. (9) provides enough signal, we simulated training in a network of 100 neurons that receives a new $\mathbf{x}$ each 150 ms. For a variety k and γ, it converged to a nearly convolutions solution within minutes (Fig. 4B, each iteration is 150 ms). Having finite inhibition did lead to worse final signal-to-noise ration ($\alpha = 10$ in Fig. 4B), but the variance of the weight was still very small. Moreover, the nature of the α-induced bias suggests that stopping training before convergence leads to better results (around 2k in Fig. 4B). See Appendix A for a discussion.

5 Experiments

We split our experiments into two parts: small-scale ones with CIFAR10, CIFAR100 [34] and TinyImageNet [35], and large-scale ones with ImageNet [36]. The former illustrates the effects of data augmentation and dynamic weight sharing on performance of locally connected networks; While the latter concentrates on dynamic weight sharing, as extensive data augmentations are too computationally expensive for large networks and datasets. We used the AdamW [37] optimizer in all runs. Our PyTorch [38] implementation is available in the supplementary material.

Datasets. CIFAR10 consists of 50k training and 10k test images of size 32×32, divided into 10 classes. CIFAR100 has the same structure, but with 100 classes. For both, we tune hyperparameters with a 45k/5k train/validation split, and train final networks on the full 50k training set. TinyImageNet consists of 100k training and 10k validation images of size 64×64, divided into 200 classes. As the test labels are not publicly available, we divided the training set into 90k/10k train/validation split, and used the 10k official validation set as test data. ImageNet consists of 1281k training images and 50k test images of different sizes, reshaped to 256 pixels in the smallest dimension. As in the case for TinyImageNet, we used the train set for a 1271k/10k train/validation split, and 50k official validation set as test data.

Networks. For CIFAR10/100 and TinyImageNet, we used CIFAR10-adapted ResNet20 from the original ResNet paper [3]. The network has three blocks, each consisting of 6 layers, with 16/32/64 channels within the block. We chose this network due to good performance on CIFAR10, and the ability to fit the corresponding locally connected network into the 8G VRAM of the GPU for large batch sizes on all three datasets. For ImageNet, we took the half-width ResNet18 (meaning 32/64/128/256 block widths), and for locally connected networks we replaced the very first layer with a convolution. This way, we were able to fit a common architecture (albeit halved in width) in the locally connected regime into 16G of GPU VRAM (after the first layer, the image resolution is reduced from 224×224 to 56×56 due to pooling followed by a strided convolution in the first block). For both networks, all layers had 3×3 receptive field (apart from a few 1×1 residual downsampling layers), meaning that weight sharing worked over 9 individual grids in each layer.

Training details. We ran the experiments on our local laboratory cluster, which consists mostly of NVIDIA GTX1080 and RTX5000 GPUs. The small-scale experiments took from 1-2 hours per run up to 40 hours (for TinyImageNet with 16 repetitions). The large-scale experiments took from 3 to 6 days on RTX5000 (the longest run was the locally connected network with weight sharing happening after every minibatch update).

5.1 Data augmentations.

For CIFAR10/100, we padded the images (padding size depended on the experiment) with mean values over the training set (such that after normalization the padded values were zero) and cropped to size 32×32. We did not use other augmentations to separate the influence of padding/random crops. For TinyImageNet, we first center-cropped the original images to size $(48 + 2\,\mathrm{pad}) \times (48 + 2\,\mathrm{pad})$ for the chosen padding size pad. The final images were then randomly cropped to 48×48. This was done to simulate the effect of padding on the number of available translations, and to compare

Regularizer	Connectivity	CIFAR10		CIFAR100				TinyImageNet			
		Top-1 accuracy (%)	Diff	Top-1 accuracy (%)	Diff	Top-5 accuracy (%)	Diff	Top-1 accuracy (%)	Diff	Top-5 accuracy (%)	Diff
-	conv	88.3	-	59.2	-	84.9	-	38.6	-	65.1	-
	LC	80.9	-7.4	49.8	-9.4	75.5	-9.4	29.6	-9.0	52.7	-12.4
Data Translation	LC - 4 reps	82.9	-5.4	52.1	-7.1	76.4	-8.5	31.9	-6.7	54.9	-10.2
	LC - 8 reps	83.8	-4.5	54.3	-5.0	77.9	-7.0	32.5	-6.1	55.7	-9.4
	LC - 16 reps	85.0	-3.3	55.9	-3.3	78.8	-6.1	34.0	-4.6	56.2	-8.8
Weight Sharing	LC - ws (1)	87.4	-0.8	58.7	-0.5	83.4	-1.6	41.6	3.0	66.1	1.1
	LC - ws (10)	85.1	-3.2	55.7	-3.6	80.9	-4.0	37.6	-1.0	62.2	-2.8
	LC - ws (100)	82.0	-6.3	52.8	-6.4	80.1	-4.8	37.1	-1.5	62.8	-2.3

Table 1: Performance of convolutional (conv) and locally connected (LC) networks for padding of 4 in the input images (mean accuracy over 5 runs). For LC, two regularization strategies were applied: repeating the same image n times with different translations (n reps) or using dynamic weight sharing every n batches (ws (n)). LC nets additionally show performance difference w.r.t. conv nets.

performance across different padding values on the images of the same size (and therefore locally connected networks of the same size). After cropping, the images were normalized using ImageNet normalization values. For all three datasets, test data was sampled without padding. For ImageNet, we used the standard augmentations. Training data was resized to 256 (smallest dimension), random cropped to 224×224, flipped horizontally with 0.5 probability, and then normalized. Test data was resized to 256, center cropped to 224 and then normalized. In all cases, data repetitions included multiple samples of the same image within a batch, keeping the total number of images in a batch fixed (e.g. for batch size 256 and 16 repetitions, that would mean 16 original images)

5.2 CIFAR10/100 and TinyImageNet

To study the effect of both data augmentation and weight sharing on performance, we ran experiments with non-augmented images (padding 0) and with different amounts of augmentations. This included padding of 4 and 8, and repetitions of 4, 8, and 16. Without augmentations, locally connected networks performed much worse than convolutional, although weight sharing improved the result a little bit (see Appendix B). For padding of 4 (mean accuracy over 5 runs Table 4, see Appendix B for max-min accuracy), increasing the number of repetitions increased the performance of locally connected networks. However, even for 16 repetitions, the improvements were small comparing to weight sharing (especially for top-5 accuracy on TinyImageNet). For weight sharing, doing it moderately often – every 10 iterations, meaning every 5120 images – did as well as 16 repetitions on CIFAR10/100. For TinyImageNet, sharing weights every 100 iterations (about every 50k images) performed much better than data augmentation.

Sharing weights after every batch performed almost as well as convolutions (and even a bit better on TinyImageNet, although the difference is small if we look at top-5 accuracy, which is a less volatile metric for 200 classes), but it is to frequent to be a plausible "sleep" phase. We include it to show that best possible performance of partial weight sharing is comparable to actual convolutions. Our results can be explained by image sizes: for larger resolutions, the network has more neurons and needs more translations for proper regularization; the spatial frequency of weight sharing depends only on the kernel size, meaning that for larger resolution more neurons end up sharing weights. Therefore, dynamic weight sharing should perform even better (relative to data augmentation) as image sizes grow.

For padding of 8, the performance did improve for all methods (including convolutions), but the relative differences had a similar trend as for padding of 4 (see Appendix B). We also trained locally connected network with one repetition, but for longer and with a much smaller learning rate to simulate the effect of data repetitions. Even for 4x-8x longer runs, the networks barely matched the performance of a 1-repetition network on standard speed (not shown).

5.3 ImageNet

On ImageNet, we did not test image repetitions due to the computational requirements (e.g., running 16 repetitions with our resources would take almost 3 months). We used the standard data

Model	Connectivity	Weight sharing frequency	ImageNet			
			Top-1 accuracy (%)	Diff	Top-5 accuracy (%)	Diff
0.5xResNet18 (1st layer conv)	conv	-	63.5	-	84.7	-
	LC	-	52.2	-11.3	75.1	-9.6
	LC	1	63.6	0.1	84.5	-0.2
	LC	10	61.6	-1.9	83.1	-1.6
	LC	100	59.1	-4.4	81.1	-3.6

Table 2: Performance of convolutional (conv) and locally connected (LC) networks on ImageNet (1 run). For LC, we also used dynamic weight sharing every n batches (ws (n)).

augmentation, which means that all networks see different crops of the same image throughout training.

Our results (Table 5) show that even infrequent dynamic weight sharing (10-100 iterations, which is 2.5k to 25k images for the used batch size of 256) achieves nearly convolutional performance. In contrast, the purely locally connected network has a large performance gap w.r.t. the convolutional one. Although we had to make the first layer of locally connected networks convolutional due to memory constraints, we do not expect it to have a large impact on performance. This is because the first layer can share a large portions of weights than the second layer (7×7 filters in the first layer result in 49 subgrids for 224^2 neurons, or 1024 neurons per subgrid; the second layer has 9 subgrids for 56^2 neurons, or about 348 neurons per subgrid).

6 Discussion

We presented two ways to circumvent the biological implausibility of weight sharing, a crucial component of convolutional networks. The first way was through data augmentation via multiple image translations. The second is dynamic weight sharing via lateral connections, which allows neurons to share weight information during a sleep-like phase; weight updates are then done using Hebbian plasticity. Data augmentations requires a large number of repetitions of the data, and, consequently longer training times, and yields only small improvements in performance. Dynamic weight sharing needs a separate "weight sharing" phase, rather than more data, and yields large performance gains. In fact, it achieves near convolutional performance even on hard tasks, such as ImageNet classification, making it a much more likely candidate than data augmentation for the brain.

There are several limitations to our implementation of dynamic weight sharing. First, it relies on precise lateral connectivity. This can be genetically encoded, or learned early on using correlations in the input data (if layer l can generate repeated patters, layer $l + 1$ can modify its lateral connectivity based on input correlations). Lateral connections do in fact exist in the visual stream, with neurons that have similar tuning curves showing strong lateral connections [39]. Second, the "sleep" phase works iteratively over layers. This can be implemented with neuromodulation that enables plasticity one layer at a time. Alternatively, weight sharing could work simultaneously in the whole network due to weight regularization (as it makes sure the final solution preserves the initial average weight), although this would require longer training due to additional noise in deep layers. Next, in our scheme the lateral connections are used only for dynamic weight sharing, and not for actual computation. However, they can be useful for computation too, as they center neural activity (in deep networks, this is done as a part of normalization layers). Finally, we trained networks using backpropagation, which is not biologically plausible [17]. However, our weight sharing scheme is independent of the main training algorithm, and therefore can be applied with any biologically plausible update rule.

Our approach to dynamic weight sharing is not only relevant to convolutions. First, it is applicable to non-convolutional networks, and in particular visual transformers [27, 28, 29, 30] (and more recent MLP-based architectures [31, 32, 33]). In such architectures, input images (and intermediate two-dimensional representations) are split into non-overlapping patches; each patch is then transformed with the *same* fully connected layer – a computation that would require weight sharing in the brain. This can be done by connecting neurons across patches that have the same relative position, and applying our weight dynamics. Second, [22] faced a problem similar to weight sharing –

weight transport (i.e., neurons not knowing their output weights) – when developing a plausible implementation of backprop. Their weight mirror algorithms used an idea similar to ours: the value of one weight was sent to another through correlations in activity.

Our study shows that both performance and the computation of convolutional networks can be reproduced in more realistic architectures. This supports convolutional networks as a model of the visual stream, and also justifies them as a "model organism" for studying learning in the visual stream (which is important partially due to their computational efficiency). While our study does not have immediate societal impacts (positive or negative), it further strengthens the role of artificial neural networks as a model of the brain. Such models can guide medical applications such as brain machine interface and neurological rehabilitation. However, that could also lead to the design of potentially harmful adversarial attacks on the brain.

References

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[2] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[4] Ossama Abdel-Hamid, Abdel-rahman Mohamed, Hui Jiang, Li Deng, Gerald Penn, and Dong Yu. Convolutional neural networks for speech recognition. *IEEE/ACM Transactions on audio, speech, and language processing*, 22(10):1533–1545, 2014.

[5] Alexis Conneau, Holger Schwenk, Loïc Barrault, and Yann Lecun. Very deep convolutional networks for text classification. *arXiv preprint arXiv:1606.01781*, 2016.

[6] Fazle Karim, Somshubra Majumdar, Houshang Darabi, and Shun Chen. Lstm fully convolutional networks for time series classification. *IEEE access*, 6:1662–1669, 2017.

[7] Grace W. Lindsay. Convolutional Neural Networks as a Model of the Visual System: Past, Present, and Future. *Journal of Cognitive Neuroscience*, pages 1–15, 02 2020.

[8] David H Hubel and Torsten N Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106–154, 1962.

[9] Kunihiko Fukushima and Sei Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pages 267–285. Springer, 1982.

[10] Daniel LK Yamins, Ha Hong, Charles F Cadieu, Ethan A Solomon, Darren Seibert, and James J DiCarlo. Performance-optimized hierarchical models predict neural responses in higher visual cortex. *Proceedings of the national academy of sciences*, 111(23):8619–8624, 2014.

[11] Seyed-Mahdi Khaligh-Razavi and Nikolaus Kriegeskorte. Deep supervised, but not unsupervised, models may explain it cortical representation. *PLoS computational biology*, 10(11):e1003915, 2014.

[12] Martin Schrimpf, Jonas Kubilius, Ha Hong, Najib J Majaj, Rishi Rajalingham, Elias B Issa, Kohitij Kar, Pouya Bashivan, Jonathan Prescott-Roy, Kailyn Schmidt, et al. Brain-score: Which artificial neural network for object recognition is most brain-like? *BioRxiv*, page 407007, 2018.

[13] Santiago A Cadena, George H Denfield, Edgar Y Walker, Leon A Gatys, Andreas S Tolias, Matthias Bethge, and Alexander S Ecker. Deep convolutional models improve predictions of macaque v1 responses to natural images. *PLoS computational biology*, 15(4):e1006897, 2019.

[14] Stephen Grossberg. Competitive learning: From interactive activation to adaptive resonance. *Cognitive science*, 11(1):23–63, 1987.

[15] Sergey Bartunov, Adam Santoro, Blake Richards, Luke Marris, Geoffrey E Hinton, and Timothy Lillicrap. Assessing the scalability of biologically-motivated deep learning algorithms and architectures. In *Advances in Neural Information Processing Systems*, pages 9368–9378, 2018.

[16] Stéphane d'Ascoli, Levent Sagun, Joan Bruna, and Giulio Biroli. Finding the needle in the haystack with convolutions: on the benefits of architectural bias. *arXiv preprint arXiv:1906.06766*, 2019.

[17] Blake A Richards, Timothy P Lillicrap, Philippe Beaudoin, Yoshua Bengio, Rafal Bogacz, Amelia Christensen, Claudia Clopath, Rui Ponte Costa, Archy de Berker, Surya Ganguli, et al. A deep learning framework for neuroscience. *Nature neuroscience*, 22(11):1761–1770, 2019.

[18] Arild Nøkland. Direct feedback alignment provides learning in deep neural networks. In *Advances in neural information processing systems*, pages 1037–1045, 2016.

[19] Theodore H Moskovitz, Ashok Litwin-Kumar, and LF Abbott. Feedback alignment in deep convolutional networks. *arXiv preprint arXiv:1812.06488*, 2018.

[20] Hesham Mostafa, Vishwajith Ramesh, and Gert Cauwenberghs. Deep supervised learning using local errors. *Frontiers in neuroscience*, 12:608, 2018.

[21] Arild Nøkland and Lars Hiller Eidnes. Training neural networks with local error signals. *arXiv preprint arXiv:1901.06656*, 2019.

[22] Mohamed Akrout, Collin Wilson, Peter Humphreys, Timothy Lillicrap, and Douglas B Tweed. Deep learning without weight transport. In *Advances in Neural Information Processing Systems*, pages 974–982, 2019.

[23] Axel Laborieux, Maxence Ernoult, Benjamin Scellier, Yoshua Bengio, Julie Grollier, and Damien Querlioz. Scaling equilibrium propagation to deep convnets by drastically reducing its gradient estimator bias. *arXiv preprint arXiv:2006.03824*, 2020.

[24] Roman Pogodin and Peter E Latham. Kernelized information bottleneck leads to biologically plausible 3-factor hebbian learning in deep networks. *arXiv preprint arXiv:2006.07123*, 2020.

[25] Behnam Neyshabur. Towards learning convolutions from scratch. *arXiv preprint arXiv:2007.13657*, 2020.

[26] Gamaleldin Elsayed, Prajit Ramachandran, Jonathon Shlens, and Simon Kornblith. Revisiting spatial invariance with low-rank local connectivity. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 2868–2879. PMLR, 13–18 Jul 2020.

[27] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[28] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European Conference on Computer Vision*, pages 213–229. Springer, 2020.

[29] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. *arXiv preprint arXiv:2012.12877*, 2020.

[30] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable detr: Deformable transformers for end-to-end object detection. *arXiv preprint arXiv:2010.04159*, 2020.

[31] Ilya Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, and Alexey Dosovitskiy. Mlp-mixer: An all-mlp architecture for vision, 2021.

[32] Hugo Touvron, Piotr Bojanowski, Mathilde Caron, Matthieu Cord, Alaaeldin El-Nouby, Edouard Grave, Armand Joulin, Gabriel Synnaeve, Jakob Verbeek, and Hervé Jégou. Resmlp: Feedforward networks for image classification with data-efficient training. *arXiv preprint arXiv:2105.03404*, 2021.

[33] Hanxiao Liu, Zihang Dai, David R So, and Quoc V Le. Pay attention to mlps. *arXiv preprint arXiv:2105.08050*, 2021.

[34] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[35] Ya Le and Xuan Yang. Tiny imagenet visual recognition challenge. *CS 231N*, 7:7, 2015.

[36] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[37] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[38] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[39] William H. Bosking, Ying Zhang, Brett Schofield, and David Fitzpatrick. Orientation selectivity and the arrangement of horizontal connections in tree shrew striate cortex. *J. Neurosci.*, 15, 1997.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 6.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See Section 6.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] See the supplementary material.
   (b) Did you include complete proofs of all theoretical results? [Yes] See the supplementary material.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See the supplementary material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See the supplementary material.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See the supplementary material; For small-scale experiments. The large-scale experiments were run 1 time due to resource constraints.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Section 5.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [No] We used publicly available datasets and open-source repositories.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] We included our implementation in the supplementary material.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [No] We used publicly available datasets.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]