Turing's Child Machine: A Deep Learning Model of Neural Development

Duncan Stothers

Advisor: Gabriel Kreiman

March 2019

Presented to the Department of Computer Science in partial fulfillment of the requirements for an A.B. degree with Honors in the Mind, Brain, Behavior track.

Harvard College

Cambridge, MA 02138

Abstract

The connection between general intelligence and development was first raised by Turing in the 1950s. Noting the incredible complexity of engineering the adult mind, he proposed instead building a child machine with a simple mind that develops into a complex adult one. The goal of building an intelligent machine, then, is refocused around engineering the neurological stages of development. It was later discovered that during development the child's brain undergoes pervasive network expansion, increasing at least an order of magnitude in size, followed by activity driven pruning - a process for which the computational role is still unknown. In the parallel world of artificial intelligence research, hand-designing deep neural networks involves architecture decisions which are often guided by intuition and trial-and-error. Architectures typically stay fixed during parameter learning, a stark contrast to the extreme architecture modifications that take place during development in a child's brain. Furthermore, it is widely understood that there is a 'small network design problem' when hand-designing deep networks. Building a smaller deep architecture that generalizes as well as a big one requires much more effort as well as more complex layers and connections. Here we model biological development using densely connected as well as convolutional deep neural networks as a means to further our understanding of neurological biological intelligence and computational artificial intelligence. Empirical results suggest the computational role of synaptic overgrowth and pruning in biology is as an unsupervised architecture search process that finds exponentially smaller architectures that generalize well. Resultant 'adult' convolutional networks that develop this way also show similarities to hand-designed networks.

Contents

1 Background

1.1 Origins

Turing first mentioned his child machine as a means to understand biological and artificial intelligence in his seminal 1950 paper, Computing Machinery and Intelligence [35]. He wrestles with the notion of formally defining intelligence, and notes that although we don't have a definition that every agrees on, we do have an example, the adult mind, which everyone agrees is intelligent. Building an adult mind as a means to create artificial intelligence he notes is difficult because of the sheer scale and complexity of it. This motivates his proposal of instead programming a simple child's mind and the small set of rules which allow it to learn and develop into an adult mind.

At the time this idea was constrained both by our lack of knowledge of neurobiological development and the computing machinery of the time. The biological revolutions of the following decades would later shed light on the detailed neurobiological stages of development, and computing power, aided most recently by the growth of GPU and parallel programming, would also exponentially improve, finally making exploring the ideas behind the Child Machine a possibility.

1.2 Development in Neurobiology

The architecture of biological neural networks is shaped by experience during 'critical periods' in early life. During critical periods the central nervous system undergoes pervasive and rapid synaptic growth followed by competitive, activity-driven pruning [13] [27] [38] [36] [4]. The result is biological neural circuitry driving an intelligent agent. These neural circuits are of interest to neuroscience because the computational role of

this overgrowth and pruning is still poorly understood.

Empirical evidence shows that experience during the critical period permanently affects the nature and performance of the adult neural circuitry [3]. Exactly why the nervous system relies on bottom-up, experience driven activity to develop rather than encoding the circuitry in a top-down genetic way is not understood, but we have reason to believe it is closely related to intelligence. In nature simpler organisms, such as the praying mantis or fruit fly, don't rely on experience driven activity to define their neural circuitry. The larger the brain and the more complex and adaptable the animal, the more it relies on experience during the critical period [23]. This connection motivates the study of neurobiological development as it relates to the study of biological and artificial intelligence.

1.3 Computer Science Motivation

The tremendous effort involved in engineering performance driven architecture designs has largely powered the recent rise of deep models for visual classification [33] [32] [18] [30] [12] [39]. Yet despite these efforts, largely guided by intuition and trial and error, the resultant networks are often over parametrized [8] [15] and contain a significant amount of redundancy in their final forms [2] [6]. These factors contribute to the data inefficiency of these large models, a key factor separating them from the intelligent vision systems we see in nature [19], and hampering practical application of them in more constrained computing environments. In many applications, hand designing better architectures is the central goal. But as much of this work is currently intuition and trial and error based, there is much room for further understanding of the underlying principles of architecture design. Understanding how nature creates its architectures could significantly advance this effort.

The on-line modification of network architecture *during training* differs notably from the current central paradigm in deep learning research of expert architecture design followed by holding the architecture constant while training. Interestingly, for a given goal on a dataset, there are usually multiple large deep networks with significant amounts of redundancy that can meet the goal, but significantly fewer small hand designed architectures that can meet the goal [15]. Additionally, it is widely known in computational neuroscience that biological networks are incredibly more energy efficient than their artificial counterparts. Thus understanding what makes small architectures generalize well is of great value to artificial intelligence research.

1.4 Related Works

Compressing artificial networks has a long history. Adding a cost to a synapse such that it decays to zero if it is less important by some measure such as the second order derivative [21] [11] [5] [10] as well as general synapse pruning [25] have been explored in the early days of artificial network design. There has been recent interest into pruning *pre-trained* deep models to increase parameter efficiency and speed [42] [31] [22]. Other techniques such as trained quantization [9] and huffman coding can further increase efficiency, and data-driven pruning performs better than using hand crafted features [14].

One approach to finding more data-efficient and parameter-efficient architectures is by taking a more hands-off approach through architecture search, and letting the data define the architecture [45]. However, the immediate major challenge with such techniques is the large computational cost [20]. The lack of differentiability of this problem motivates alternative approaches such as genetic algorithms, utilized to good effect by [24] in 1989 on small networks. But scaled up to deeper

models the computational complexity of evolutionary methods becomes immense [28]. Another strategy is to make the discrete space differentiable, which Shin *et al* showed can be used to increase efficiency [29]. However, most recent methods define some form of an architect or controller network which learns to build models for a given dataset [43] [26] [44]. The computational challenge with these techniques is that they often involve training many different models from scratch, in addition to the controller, to learn the subtleties between architecture and performance making them computationally arduous. This high computational cost limits practical applications, and the black box nature of these approaches limits their contributions to our understating of architecture design.

2 Dense Network Model

2.1 General Process

In our first model we aim to model biology as closely and as simply as possible, so we create a simple process that models synaptic expansion and pruning and apply it to a multi-layer perception. The goal with the dense network model is to identify whether the expansion and pruning of a network exhibits architecture search properties.

In developing neurobiological hardware, architectural changes take place concomitant with network training. The first phase is dominated by random synaptic expansion of the network, defining an exploration phase through architecture space, starting at a sparse random network and ending at peak synaptic density. At peak synaptic density the developmental process becomes dominated by the pruning of synapses, and total synaptic density starts to fall. In the end, the model weights

encode information to process the task as usual, but the network itself also encodes information about the task because it was pruned in a data-driven way. This developmental process can be modeled stochastically by a simple ϵ-greedy process during training. We define an initial architecture A_0 and weights W_0 for our network where A_0 is a randomly initialized sparse network, and W_0 are randomly initialized. At each training step i, in addition to updating weights, the architecture is updated to A_i: either randomly expanding the network or pruning the network by activity. The decision d on whether to grow or prune is controlled by flipping a biased coin $d \sim Bern(\epsilon_i)$ where $Bern$ refers to standard Bernoulli distribution sample, ϵ_i controls the bias at iteration i and $\epsilon_0 = 1$.

$$A_i := \begin{cases} \text{RandExpand}(A_{i-1}) & d = 1 \\ \text{ActivityPrune}(A_{i-1}, W_{i-1}) & d = 0 \end{cases}$$

Implementations of $\text{RandExpand}(A_{i-1})$ and $\text{ActivityPrune}(A_{i-1}, W_{i-1})$ depend on the architecture space and type of network, but we can define a general hyper parameter γ and enforce that $|\text{RandExpand}(A_i)| = |A_i| + \gamma$ and symmetrically $|\text{ActivityPrune}(A_i)| = |A_i| - \gamma$, where $|A_i|$ refers to architecture size. γ then controls the max size our network is expected to reach during the search as well as the coarseness/fineness of our adjustments to the architecture between iterations. We also define *a priori* how many iterations n we want the search/training to take place over, the initial size of the randomly initialized architecture N. At each iteration $\epsilon_i := \epsilon_{i-1} - 1/n$ ensuring that at iteration n, $E[|A_n|] = |A_0|$ where $|A|$ refers to the size of the network, measured in alive synapses, neurons, or synaptic density depending on the application. Overall this defines a

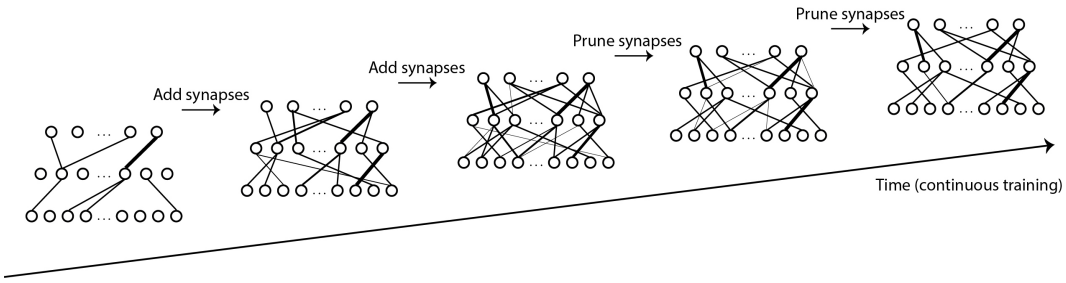

Figure 1: **Schematic illustration of synaptic search model.** In traditional architectures, the number of trainable non-zero parameters (synapses) are hard wired from the onset and the weights of those synapses are allowed to change via training. In contrast, here the number of synapses in the network and the synaptic weights both change simultaneously during the training process so both parameter learning and architecture learning take place.

training process such that initially total network size increases in a randomized exploratory manor, then decreases in a greedy exploitation manner. Because the process is stochastic, the transition is smooth, especially for small γ and $1/n < \gamma$.

2.2 MLP Algorithm

To apply the general process to a dense network or multi-layer perceptron, we define a fixed set of layers $1, ..., m$ and units per layer $L_1, ..., L_m$ which we do not change. Thus the architecture search takes place over subgraphs of the graph defined by dense connections between units L_i and L_{i+1}, for $1 \leq i < m$. It is possible to consider a larger search space which allows the number of layers to change or units per layer to change; for simplicity, we do not include these possibilities here. To keep track of whether a synapse between two neurons is 'alive', for each weight matrix $W_1, ..., W_{m-1}$ we have a table $D_1, ..., D_{m-1}$ of the same shape. Synapse $W_i[j, k]$ is considered alive and part of the graph (and thus allowed to change as usual) between training iterations if $D_i[j, k] = 0$. Synapse $W_i[j, k]$ is considered dead and

not part of the graph if $D_i[j,k] = 1$ and $W_i[j,k] = 0$. For a given γ we also need to define the process by which we add or remove synapses from the network. We interpret γ as a percent of total synaptic density in this case. To RandExpand the network we perform the following: for each W_i we sample $\lceil \gamma * (L_i * L_{i+1} + L_{i+1}) \rceil$, (j,k) indices from D_i where $D_i[j,k] = 1$, without replacement and set $D_i[j,k] = 0$ so that during subsequent training steps these weights can change. To ActivityPrune, for each layer we sort the weights by their absolute value, set the smallest $\lceil \gamma * (L_i * L_{i+1} + L_{i+1}) \rceil$ weights to zero, and set their value in the corresponding D table to 1 so that they stay fixed at zero between training steps. Note that when we seek to increase or decrease synaptic density by γ percent we do this layer by layer, reducing the variance with respect to a situation where γ percent were chosen simultaneously between layers. We randomly initialize the network by picking a starting synaptic density N and for each weight matrix set $\lceil (1 - N) * (L_i * L_{i+1} + L_{i+1}) \rceil$ weights at random to zero and their corresponding values in D to 1. The rest of the alive weights are randomly initialized.

2.3 Developmental Model

Figure two describes our model of the synaptic overgrowth and pruning stages of development using a multi-layer perceptron / fully connected network, trained on the MNIST handwritten digit classification task, with $m = 3$ layers, 784 units in the input layer, 16 units in layer 2 and a final classification layer with 10 units. Iterative model creation / deletion was implemented with Tensorflow's [1] Keras library [7]. For the dense model experiments, the training loop was rewritten to ensure dead weights stayed fixed at zero between iterations, and within each iteration one gradient step was taken over a 2048 minibatch.

The process of synaptic growth and synaptic pruning is illustrated in

Algorithm 1 Dense Critical Period Model

procedure SynapseSearch(γ, n, N, X, Y)
  $\epsilon \leftarrow 1$
  $d \leftarrow 1$
  $\gamma \leftarrow \gamma/100$ ▷ We interpret Gamma as a percent
  Initialize $D_1, ..., D_{m-1}$
  Initialize $W_1, ..., W_{m-1}$
  for $i = 1, ..., \lceil n + N/\gamma \rceil$ **do**
    $d \leftarrow Bern(\epsilon)$
    if $d = 1$ **then**
      for $l = 1, ..., n-1$ **do**
        Randomly Change $\lceil \gamma * (L_i * L_{i+1} + L_{i+1}) \rceil$ values in D_l from 1 to 0
    else
      for $l = 1, ..., n-1$ **do**
        Sort alive synapses by $|W_i[j,k]|$
        for The weakest $\lceil \gamma * (L_i * L_{i+1} + L_{i+1}) \rceil$ synapses **do**
          $W_l[j,k] \leftarrow 0$
          $D_l[j,k] \leftarrow 1$
    $\{W\}_1^{m-1} \leftarrow$ TrainStep($\{W\}_1^{m-1}, \{D\}_1^{m-1}, X, Y$)
    $\epsilon \leftarrow \max(0, \epsilon - 1/n)$
  return $\{W\}_1^{m-1}, \{D\}_1^{m-1}$

Figure 2a and performance is reported in Figure 2b, as a function of the number gradient/training steps. For MNIST in particular, the results indicate that a very small network (up to a synaptic density of $\tilde{5}\%$), can achieve comparable accuracy to the full, densely connected network, even though this network has 95 percent fewer parameters and therefore has much higher efficiency.

2.4 Architecture Search Properties

Key to the algorithm's ability to find parameter-efficient solutions is the architecture search process it carries out. To isolate and illustrate this phenomenon, the algorithm was run on the MNIST dataset and the architectures at 1,2,3,4,5,10,15, and 20 percent synaptic density were

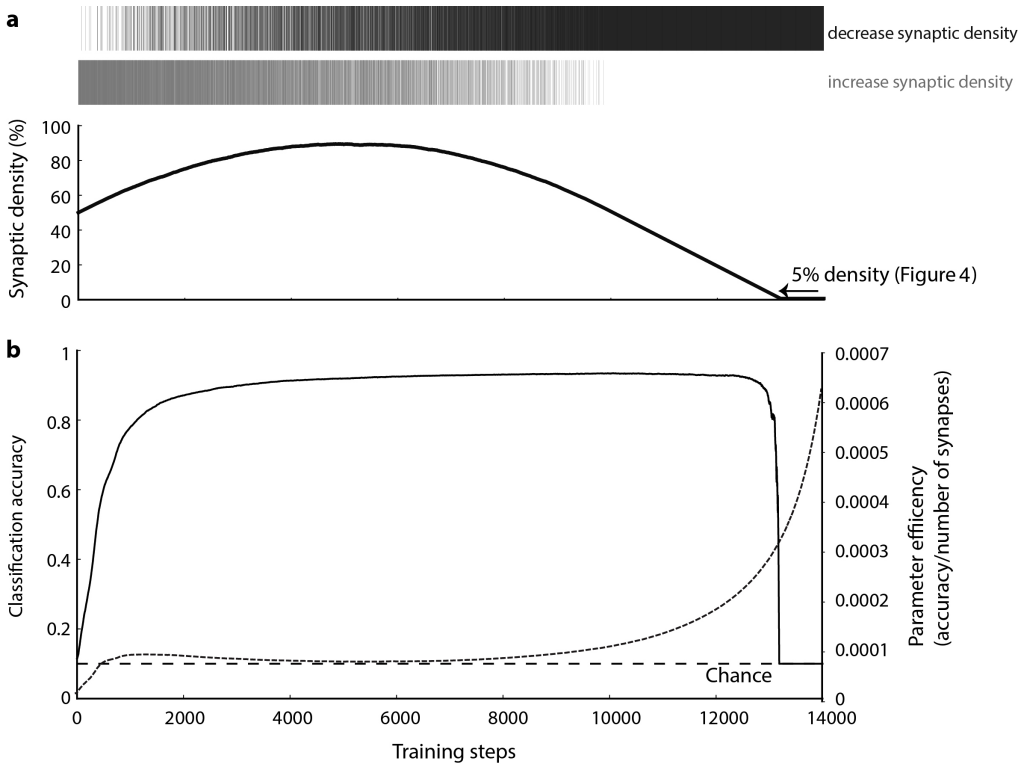

Figure 2: **Dense network critical period model for a 784x16x10, fully-connected MLP** with $n = 10000$, $\gamma = 0.016$,$N = 50$ At each step, synapses were either added (gray vertical bars) or removed (black vertical bars) in a probabilistic fashion, leading to the synaptic density change shown below. b. Classification test accuracy and efficiency as a function of training steps. Chance = 10 percent (dashed line)

saved. The weights for these biologically plausible architectures were then re-initialized, trained from scratch (Figure 3 b), and the final test accuracy after 10000 gradient steps on a batch size of 2048 is plotted as the green points in Figure 3c. We compared these biological architectures to random architectures with the same number of free parameters/synapses which were also trained from scratch (Figure 3a) under the same conditions with final test accuracy plotted as the blue dots (Figure 3b).

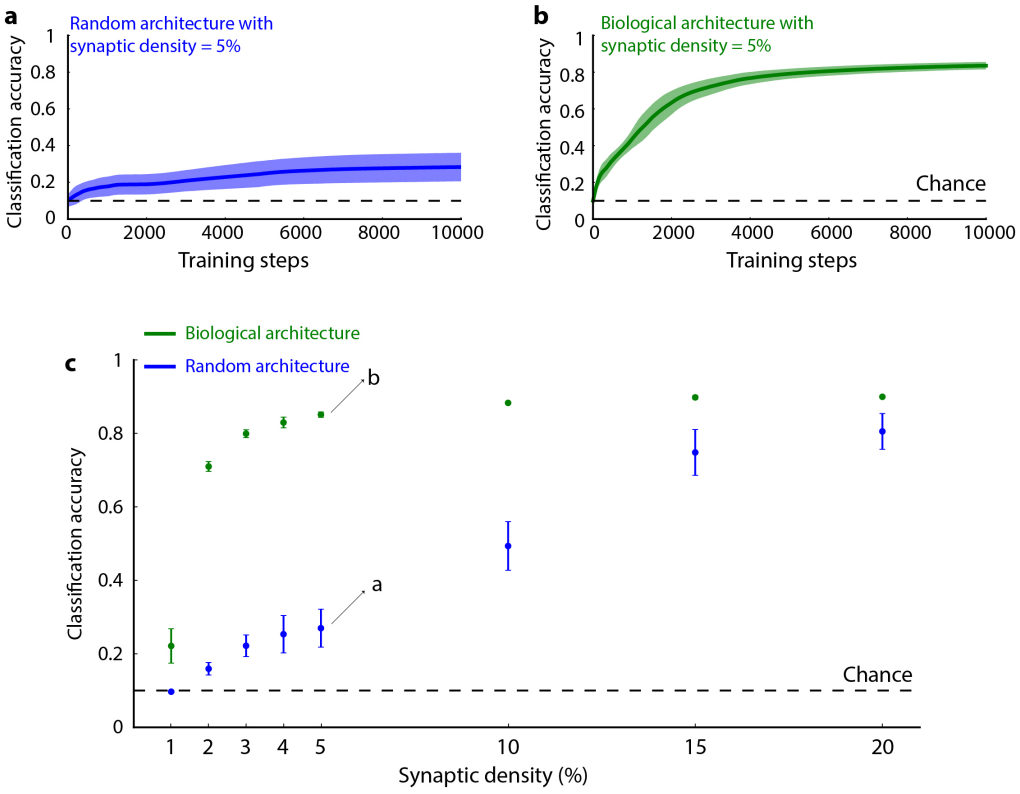

Figure 3: **The biological architecture found during the dense critical period shows higher performance than a random architecture.** a. A random architecture was generated such that the overall density of connections was 5 percent. The network was trained from scratch until convergence. b. Following the procedure schematically illustrated in Figure 1, the number of synapses in the network grew and was then pruned up to a synaptic density of 5 percent (arrow in Figure 3a), thus learning a biological architecture. The weights in this network were then randomly initialized and the network was re-trained. Shaded areas represent one standard deviation (n=10 iterations). c. The biological architecture showed higher classification accuracy than random architectures for all synaptic densities up to 20 percent (t-test, $p < 0.01$). The arrows point to the examples shown in parts a and b.

2.5 Transfer Learning Properties

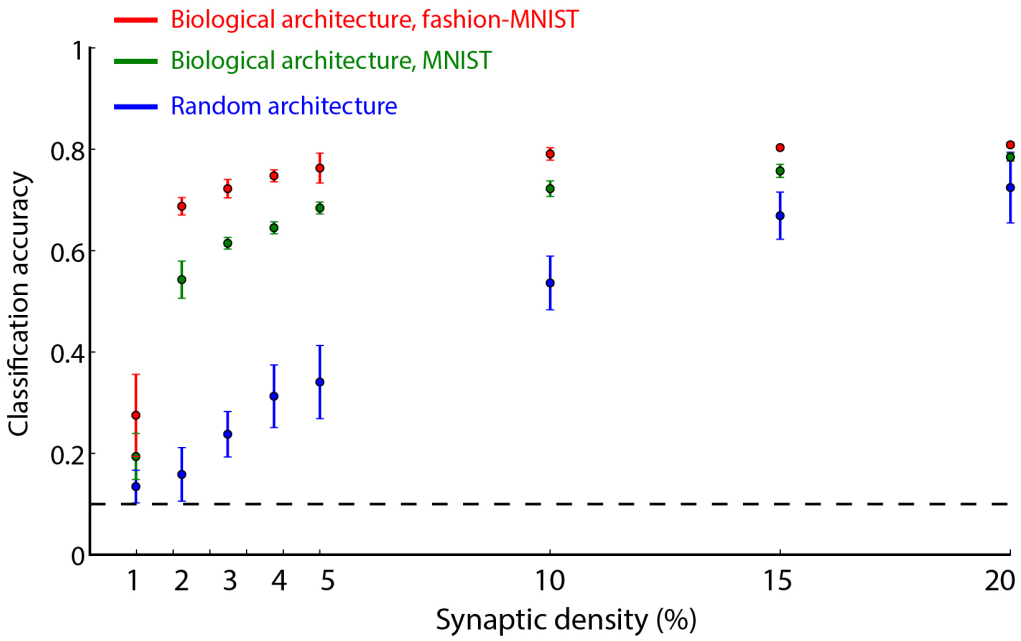

Figure 4: **Biological architectures can thrive through transfer learning.** Pictured is performance on the fashion-MNIST dataset [40]. In red, we considered a biological architecture that was grown and pruned using the fashion-MNIST and in Green, grown and pruned with the MNIST dataset. n=5 (pairwise t-test, for density = 1,20 $p < 0.1$, all others $p < 0.005$)

The efficiency of the biological architectures motivates the question of how much information about the data is actually encoded in the architecture, and how the brain could utilize this architecture level encoding of information for other tasks through architecture-level transfer learning. To study this effect of transfer learning through architecture we consider a classification task similar to MNIST, fashion-MNIST, for classification of 28x28 articles of clothing into 10 categories. Architectures are searched for using our dense critical period model of developmental training and then weights are re-initialized and the networks are retrained as before in Figure 4, but this time they are trained on fashion-MNIST

(Figure 6, green points) even though they were being trained on MNIST during the dense critical period training. They are compared to biological architectures found by dense critical period training on fashion-MNIST (Figure 6, red points). Both architectures show a higher performance for a given synaptic density than random architectures, and specifically the green points show significantly higher performance than random. This test shows how transfer-learning can take place on the architecture level, and models how different but related experience during the critical period can be used to learn new tasks later in life.

2.6 Data Efficiency

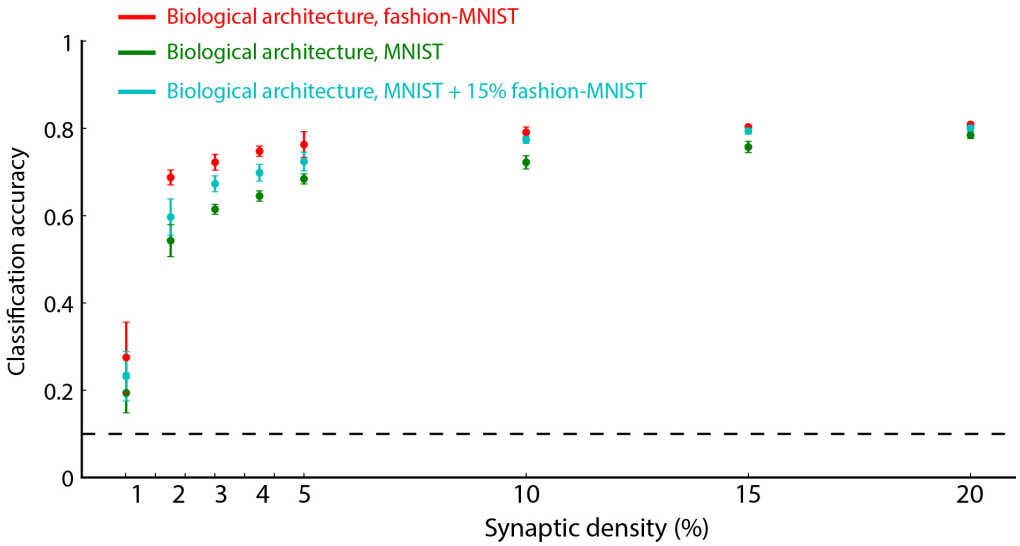

Figure 5: **Minimal exposure to stimuli during development can subsequently help with transfer learning.** In the cyan dots, the network was exposed to 15 percent fashion-MNIST during the dense critical period training which led to a significant architectural improvements when retrained on fashion-MNIST (pairwise t-test, $p < 0.05$)

An intriguing aspect of nervous system development is that during critical periods in development even minimal exposure to new tasks and information can lead to significant improvement in performance years

later and, conversely, limited exposure to new tasks can lead to severe impairment in subsequent ability to learn those tasks. For example, rearing an animal in the dark [37], or not exposing the animal to specific stimuli such as faces during those critical periods, significantly alters the development of the visual system [3]. Here we directly tested this idea in our model of development.

Interestingly, when only 15 percent of the fashion-MNIST data is added to the MNIST data during the dense critical period training, the performance through transfer learning rises significantly. This implies that the information encoded in the architecture is sensitive to the nuances in the data, and that the network architectures are not simply dominated by the structure of the most frequent classes. The relatively large influence of a very small amount of data during the developmental process on the networks capacity to learn the task post development corroborates biological plausibility [16] [17].

2.7 Discussion

The process of exploration dominated by synaptic growth for dense networks is marked by increasing performance, and the exploitation phase dominated by synaptic pruning is marked by increasing efficiency. We explore the data-driven architecture search properties of the dense critical period model training. Empirically, we demonstrate that this model of the critical period finds small network architectures that perform significantly better than their random counterparts. Our results motivate a biologically plausible strategy discovered by the nervous system through evolution to be more data and computationally efficient by exploiting architecture search properties while developing. To further analyze how information is encoded in the architecture we demonstrate a novel method for transfer learning on the architecture level, and show how it is sensitive

to relatively small amounts of data during the search/pruning phase. This demonstrates a biologically plausible method the central nervous system could potentially exploit for more data-efficient learning of new tasks.

These results underscore the idea that the critical period during which synaptic overgrowth and pruning takes place is a biological architecture search driven by experience. The results also suggest that the process is general does significantly outperform random architectures. However, dense networks are likely not our best model for the brain and the MNIST and fashion-MNIST tasks are significantly easier than those encountered by people everyday. In the next section we expand these results by modelling the critical period with much larger deep convolutional neural networks on the more challenging CIFAR-10 image recognition task.

3 Convolutional Network Model

3.1 Setup

The base architecture is a modified version of Alexnet [18] adapted for the CIFAR-10 image recognition problem [41]. The CIFAR-10 training set consists of 50000 (32,32,3) dimensional images labelled as one of ten classes with 100000 images in the validation set. It is significantly more challenging than MNIST because high generalization accuracy requires true object detection instead of simple pattern recognition, so is the most common benchmark to test deep learning computer vision algorithms.

The standard architecture adapted for CIFAR-10 is 64 5x5 convolutions, 3x3 max pool, batch normalization, 256 5x5 convolutions, 3x3 max pool, batch normalization, flatten, followed by three dense layers with 382, 194, and 10 units with a softmax on the output. The convolutionl layers used are 'double for loop' convolutions from [34], which allow greater access to

the convolutional operation compared to Tensorflow's built in convolution operation. All layers contain rectified linear nonlinearities and categorical cross-entropy loss is used. Unless otherwise stated, the optimizer used is standard stochastic gradient descent with a batch size of 128 and with a learning rate schedule of 0.1 for 0 to 100, 0.01 for 100 to 200, and 0.001 for 200 to 250.

The following experiments were conducted on the GPU partition of the Harvard Medical School Orchestra 2 (O2) super-cluster, always with 1 CPU and GPU and between 5 - 16 gigabytes of memory. To train the network once from scratch for 250 epochs takes between 5 and 10 hours depending on the particular GPU. Jobs were either completed in interactive jupyter notebook sessions or were submitted, typically overnight, as batch jobs to the SLURM scheduler.

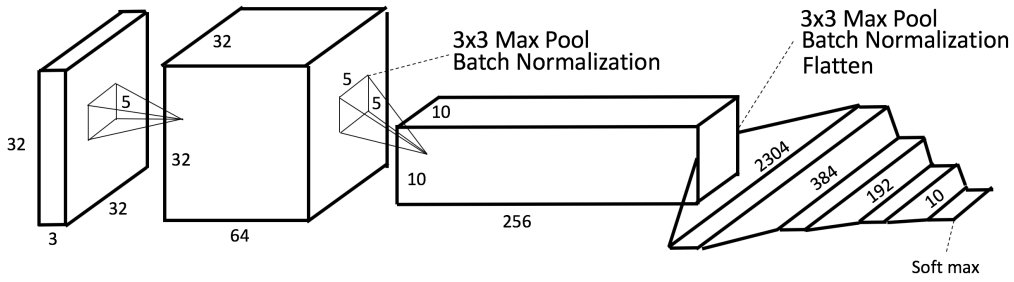

Figure 6: **Alexnet style architecture for CIFAR-10** The base architecture used for the convolution experiments - an adaptation of the original Imagenet Alexnet for the smaller CIFAR-10 dataset.

3.2 Pruning Kernels

In the dense networks experiments individual synapses were pruned during the ActivityPrune steps and the absolute value of the weight is used as a proxy for activity. In these experiments convolutional kernels are pruned and the L2 norm is used as a proxy for activity. Figure 7

compares randomly pruning kernels from the model versus pruning the kernels with the least activity measured by the lowest L2 norm of the weights in the kernel. As expected random pruning does roughly linear damage to performance. Kernels are pruned from either the first or the second convolutional layers, and whenever a kernel is pruned the corresponding connections to the first of the three dense layers are also removed/pruned. Other than this pruning in the first dense layer there is no other pruning in the dense layers. This was chosen to isolate the effect of architecture search of the distribution of kernels between layers one and two. Figure 8 compares activity pruning without retraining to with retraining 15 epochs per prune at a 0.001 learning rate. The logarithmic shape suggests that a significant fraction of the convolution kernels can be pruned away while doing minimal hard to the network, similar to pruning individual synapses in a dense network with retraining.

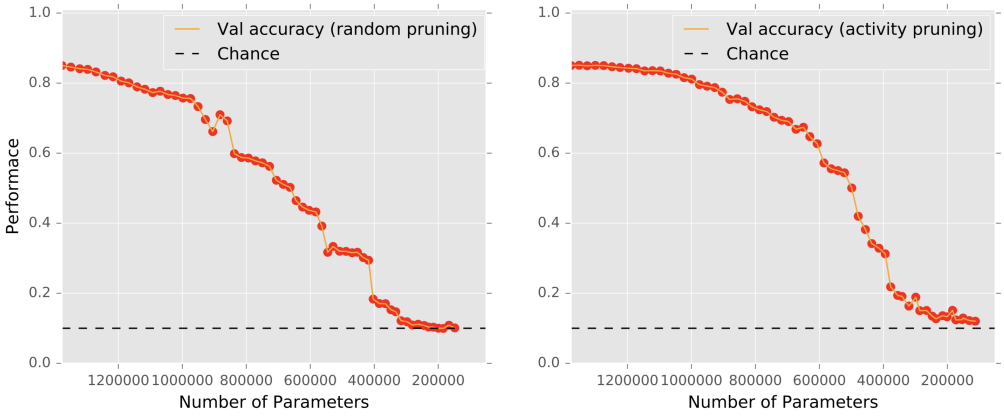

Figure 7: **Activity based pruning vs random pruning** Mini Alexnet trained on CIFAR-10, then accuracy is plotted as kernels are pruned away either randomly or by minimum activity. Random pruning does roughly linear damage but activity pruning gives a more concave curve.

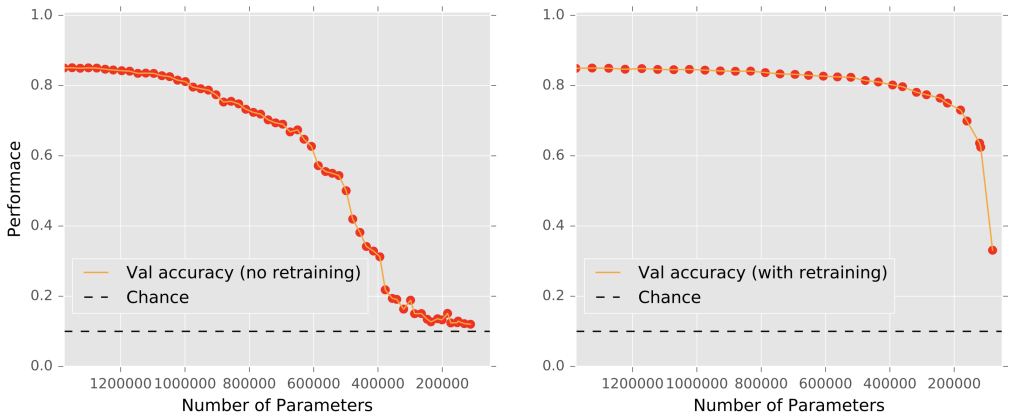

Figure 8: **Activity based pruning with and without retraining** Activity pruning with 15 epoch retraining at 0.001 learning rate between prunes significantly helps preserve performance.

3.3 Developmental Model

The convolutional model of the critical period has a number of key differences from the dense network model. First, instead of adding a network expansion step we instead start with a large network with 64 kernels in layer 1 and 256 in layer 2, for simplicity. We assume that in the expansion phase the random network growth leads to a generically large connected network. Once the large network is trained for 250 epochs we enter the pruning phase which repeats the process of removing the 10 least active kernels and training for an additional 15 epochs. This process is what creates the step pattern in figure 9. We prune multiple kernels at a time to reduce the computation time, as the prune steps have a large computational overhead. The GPU memory is cleared, the pruning algorithm takes place on the CPU, then a new model and weights needs to be sent to the GPU. In our model of the critical period we stop pruning once we start seeing validation accuracy drops. The key insight is the exponential decrease in the size of the model, which can be seen in the straight decreasing 'architecture size' line versus the right y axis

which is on a log scale.

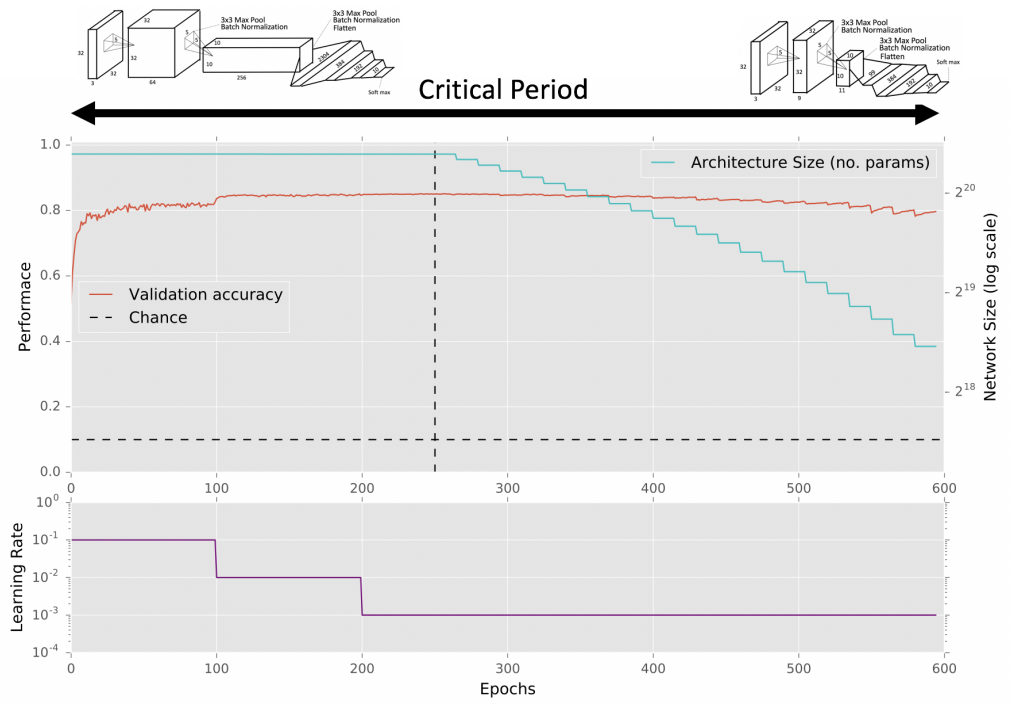

Figure 9: **Convolutional critical period model** Model of the critical period using Alexnet. The first 250 epochs are 'standard training' of a big model with 64 and 256 kernels in layers one and two. At 250 epochs we repeat prune the 10 least active kernels and train for an additional 15 epochs with a 0.001 learning rate. The model size decreases exponentially as kernels are removed.

3.4 Architecture Search Properties

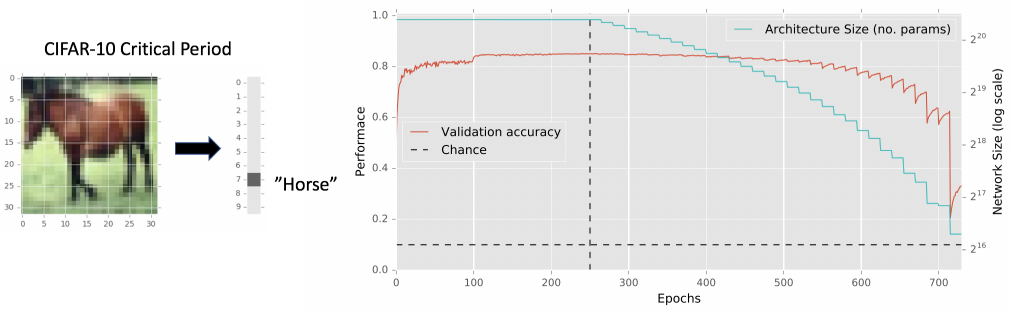

Figure 10: **Full pruning of CIFAR-10 critical period** To test the architecture search properties of the convolutional critical period model, we prune the network to close to chance accuracy. The process is the same as in figure 9 but we continue to prune instead of stopping early to preserve accuracy. Architectures with 300, 200, 100, 30, and 20 total kernels are saved for use in figure 11.

Similar to the dense network experiments, we would like to examine the architecture search properties of this procedure. We let the process prune the network to close to chance accuracy, and save the architectures with 300,200,100,30 and 20 kernels. The weights of these networks are then reinitialized and trained from scratch and plotted in figure 11 verses random architectures of the same size. E.g. if the 300 size CIFAR architecture has 50 kernels in layer one and 250 in layer two, the random architecture randomly splits 300 kernels between layer one and two. The specific part of the architecture the model searches for is the ratio of kernels in layer one to layer two.

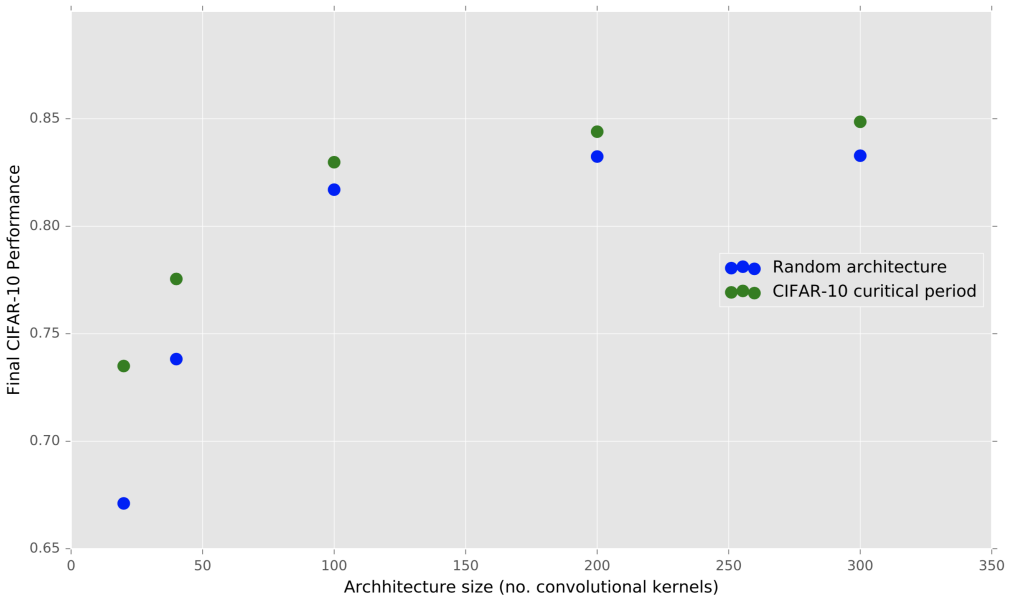

Figure 11: **The convolutional critical period architectures have higher performance than random ones** random architectures were sampled and trained three times for each size, and the final performance shown is the average of the three final performances. Similar to the dense model, we can see the pruning process has a positive search effect on architecture which is most pronounced at small architecture sizes. The difference in absolute validation accuracy is not as large as in the dense experiments, because the process of using kernels is significantly less fine grained that pruning individual synapses.

The above results show the architecture search ability of the pruning process, but what if it is simply selecting architectures that are arbitrarily good at memorization instead of generalizeable learning? To test this we introduce another critical period with a network trained on pure noise inputs and outputs (Figure 12). The learning rate for epochs 0-20 is constant at 0.01, which drops to 0.001 after epoch 20. When this network is pruned down the parts which are least useful for memorizing pure noise are pruned away, maximizing the networks pure memorization ability for data with no hierarchical patterns or features. The random inputs are mean and standard deviation normalized, the same as the CIFAR inputs.

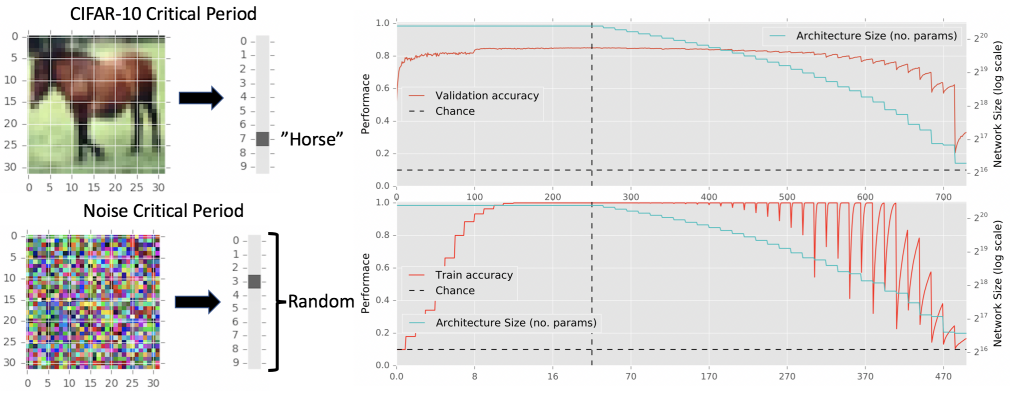

Figure 12: **Full pruning of CIFAR-10 critical period and noise critical period**

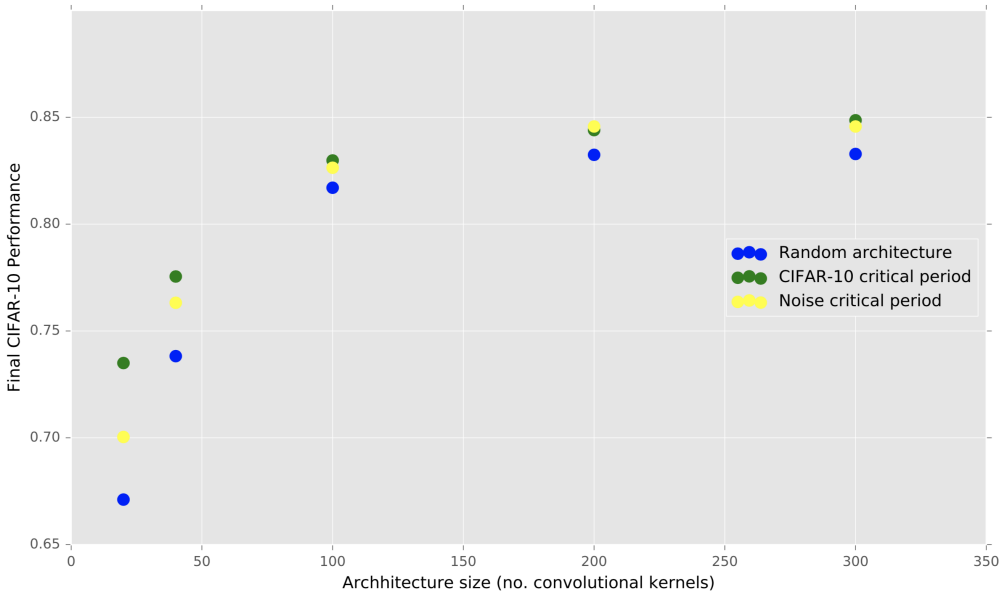

Figure 13: **The noise critical period architectures are better than random ones, but do not perform as well as CIFAR-10 critical period ones** The architectures exposed to noise during the critical period close the gap for large architecture sizes 200 and above, but still under perform for the smallest architectures. This shows the architecture search ability of the model is not simply finding the model which is best at memorizing structure-less inputs to outputs. The structure of the data matters to the architectures searched.

When we examine the architectures learned in the noise and CIFAR-10

critical periods (figure 14), we can see the CIFAR-10 critical period selects more kernels in the second layer, whereas the noise critical period learns an architecture with more kernels in the first layer. It is common practice in hand designing architectures to have more kernels in the later layers because this is where higher order features that are richly found in natural images are computed upon. The noise inputs and outputs lack these rich image statistics so for the purpose of memorizing specific inputs to outputs it is more useful to have kernels in the first layer.

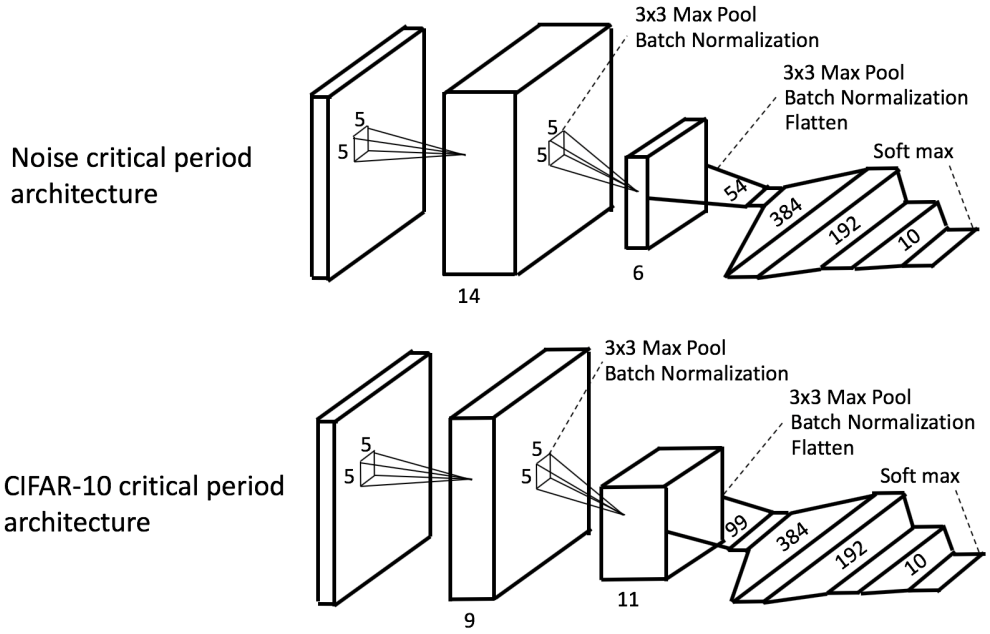

Figure 14: **Exposure to natural images during the critical period leads to architectures with more density in the later layers** Exposure to hierarchical features (edges to lines to shapes, etc.) during the critical period leads to networks with more kernels later in the network which are needed to compute on these higher order features. The noise critical period has no need to operate on higher order features so keeps all its kernels in the first layer

Figure 10 suggests that during the critical period the architecture is optimized around image statistics. With richer image statistics that include edges, lines, shapes, etc. the architecture learned has more kernels in the

part of the network which deals with higher order image statistics. But this begs the question how the labels of the images affect the architecture. To answer this question a third critical period is modelled but where the label for each CIFAR-10 image is assigned randomly. Thus the correlation between the inputs and outputs becomes zero like the noise case and the task reduces to simply memorizing which number to assign to each input, but the inputs are still natural images with rich image statistics.

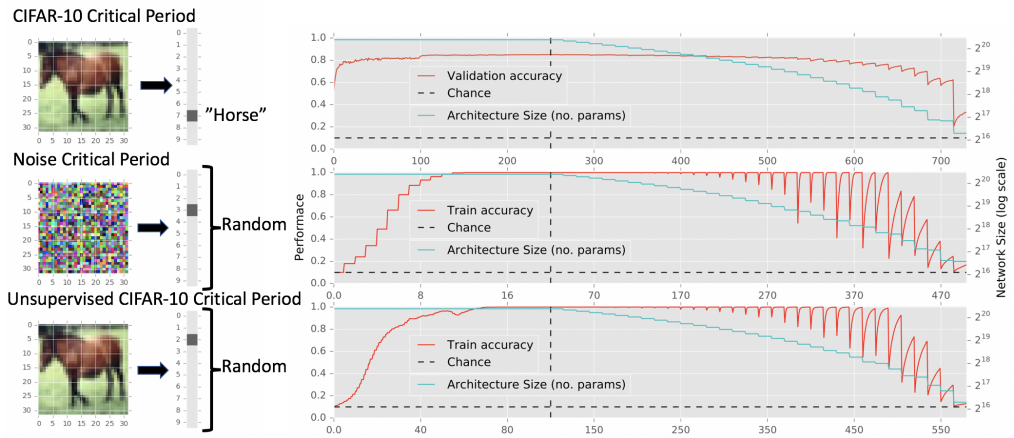

Figure 15: **Full pruning of CIFAR-10 critical period, noise critical period, and unsupervised CIFAR-10 critical period** Each image in the train set is assigned a random label. The network this trains on this data, learning to memorize inputs. The pruning process selects the small architectures that generalize well for memorizing natural images

Surprisingly, the architectures from the unsupervised CIFAR-10 critical period generalize just as well or as the architectures from the supervised CIFAR-10 critical period (figure 16). The nature of the randomized label task forces the network to memorize each image because the output label is uncorrelated with the input. If the label is correlated with the input (as it is in the supervised case) then the network can avoid memorizing each image by learning the natural mapping. We hypothesize that to memorize each image in the unsupervised case, the most efficient way to do this is to learn the natural image statistics. Thus the network the pruning process

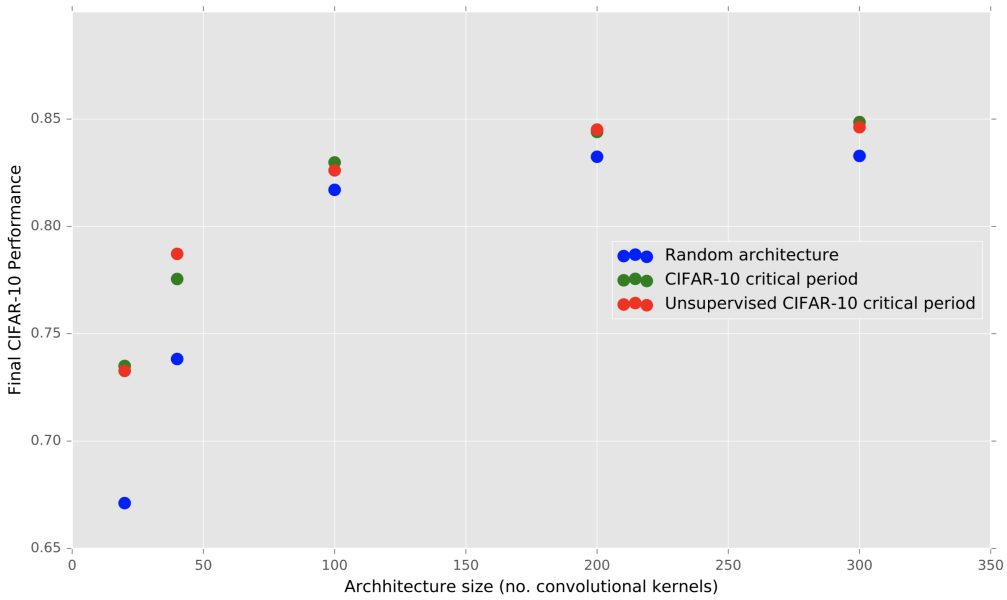

Figure 16: **The unsupervised critical period does just as well as the supervised critical period** Surprisingly, when the architectures from the unsupervised critical period are retrained on normal CIFAR-10 they generalize just as well as architectures exposed to the full supervised CIFAR-10 during the critical period.

finds has many kernels in the second convolutional layer to fully exploit the rich image statistics of natural images as can be seen in figure one. In fact, the unsupervised architecture is almost the same as the supervised case for every size except 30, where it outperforms the supervised CIFAR architecture by having almost double the kernels in the second layer as the first. This result helps explain why a child's brain can learn an efficient architecture during the critical period without labelled object data. Simply remembering what one sees without doing object detection is enough for an exponentially small generalizeable architecture to develop during the critical period.

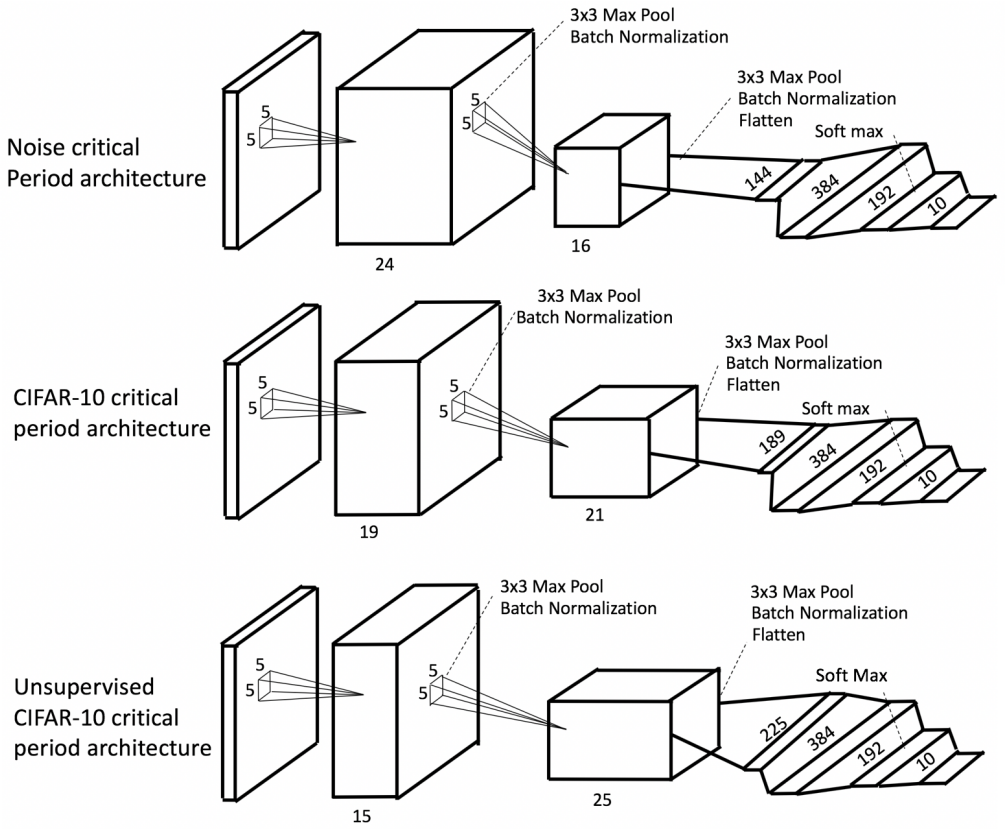

Figure 17: **The unsupervised critical period network has a similar or better architecture than the supervised critical period network** Having double or more kernels as layers deepen is a common hand-designed architecture choice

3.5 Discussion

The data suggest that the critical period evolved as an unsupervised architecture search process which finds exponentially small architectures that generalize well. The findings explain why visual development (unlike language development) is independent of specific parent supervision and/or labelled object data. The final architectures from the supervised CIFAR-10 critical period and unsupervised CIFAR-10 critical period resemble hand designed architectures in that they distribute more kernels to the later layers to take advantage of the image statistics that arise in

the network at these later levels. The exponential compression in the architecture size also helps explain the efficiency of biological networks compared to artificial ones. Practically, this procedure could be used to find small architectures for tasks where only a small amount of labelled data is available. A large generic architecture could memorize random labels for large databases of unsupervised data, and the resultant small efficient architecture would be selected such that it fits the statistics of the data well. This architecture could then be used to learn from the limited supervised data.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] J. M. Alvarez and M. Salzmann. Learning the number of neurons in deep networks. In *Advances in Neural Information Processing Systems*, pages 2270–2278, 2016.

[3] M. J. Arcaro, P. F. Schade, J. L. Vincent, C. R. Ponce, and M. S. Livingstone. Seeing faces is necessary for face-domain formation. *Nature neuroscience*, 20(10):1404, 2017.

[4] N. Berardi, T. Pizzorusso, and L. Maffei. Critical periods during sensory development. *Current opinion in neurobiology*, 10(1):138–145, 2000.

[5] Y. Chauvin. A back-propagation algorithm with optimal use of hidden units. In *Advances in neural information processing systems*, pages 519–526, 1989.

[6] Y. Cheng, F. X. Yu, R. S. Feris, S. Kumar, A. Choudhary, and S.-F. Chang. An exploration of parameter redundancy in deep networks with circulant projections. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2857–2865, 2015.

[7] F. Chollet et al. Keras. https://keras.io, 2015.

[8] M. Denil, B. Shakibi, L. Dinh, N. De Freitas, et al. Predicting parameters in deep learning. In *Advances in neural information processing systems*, pages 2148–2156, 2013.

[9] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[10] S. J. Hanson and L. Y. Pratt. Comparing biases for minimal network construction with back-propagation. In *Advances in neural information processing systems*, pages 177–185, 1989.

[11] B. Hassibi, D. G. Stork, and G. Wolff. Optimal brain surgeon: Extensions and performance comparisons. In *Advances in neural information processing systems*, pages 263–270, 1994.

[12] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[13] T. K. Hensch. Critical period plasticity in local cortical circuits. *Nature Reviews Neuroscience*, 6(11):877, 2005.

[14] Q. Huang, K. Zhou, S. You, and U. Neumann. Learning to prune filters in convolutional neural networks. *arXiv preprint arXiv:1801.07365*, 2018.

[15] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[16] E. I. Knudsen and P. F. Knudsen. Sensitive and critical periods for visual calibration of sound localization by barn owls. *Journal of Neuroscience*, 10(1):222–232, 1990.

[17] E. I. Knudsen and M. Konishi. A neural map of auditory space in the owl. *Science*, 200(4343):795–797, 1978.

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[19] B. M. Lake, T. D. Ullman, J. B. Tenenbaum, and S. J. Gershman. Building machines that learn and think like people. *Behavioral and Brain Sciences*, 40, 2017.

[20] B. Z. Q. Le and B. Zoph. Using machine learning to explore neural network architecture. *Google Research Blog*, 2017.

[21] Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990.

[22] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

[23] J. Lichtman. Mcb 80: Lectures on development. 2018.

[24] G. F. Miller, P. M. Todd, and S. U. Hegde. Designing neural networks using genetic algorithms. In *ICGA*, volume 89, pages 379–384, 1989.

[25] M. C. Mozer and P. Smolensky. Skeletonization: A technique for trimming the fat from a network via relevance assessment. In *Advances in neural information processing systems*, pages 107–115, 1989.

[26] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018.

[27] D. Purves and J. W. Lichtman. Elimination of synapses in the developing nervous system. *Science*, 210(4466):153–157, 1980.

[28] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. Le, and A. Kurakin. Large-scale evolution of image classifiers. *arXiv preprint arXiv:1703.01041*, 2017.

[29] R. Shin, C. Packer, and D. Song. Differentiable neural network architecture search. 2018.

[30] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[31] S. Srinivas and R. V. Babu. Data-free parameter pruning for deep neural networks. *arXiv preprint arXiv:1507.06149*, 2015.

[32] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, et al. Going deeper with convolutions. Cvpr, 2015.

[33] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.

[34] K. Ta Preechakul. 2d convolution layer without conv2d (in keras).

[35] A. M. Turing. Computing machinery and intelligence. In *Parsing the Turing Test*, pages 23–65. Springer, 2009.

[36] T. N. Wiesel and D. H. Hubel. Single-cell responses in striate cortex of kittens deprived of vision in one eye. *Journal of neurophysiology*, 26(6):1003–1017, 1963.

[37] T. N. Wiesel and D. H. Hubel. Comparison of the effects of unilateral and bilateral eye closure on cortical unit responses in kittens. *Journal of neurophysiology*, 28(6):1029–1040, 1965.

[38] D. J. Willshaw and C. Von Der Malsburg. How patterned neural connections can be set up by self-organization. *Proc. R. Soc. Lond. B*, 194(1117):431–445, 1976.

[39] K. Wu, E. Wu, and G. Kreiman. Learning scene gist with convolutional neural networks to improve object recognition. *arXiv preprint arXiv:1803.01967*, 2018.

[40] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[41] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

[42] H. Zhou, J. M. Alvarez, and F. Porikli. Less is more: Towards compact cnns. In *European Conference on Computer Vision*, pages 662–677. Springer, 2016.

[43] Y. Zhou and G. Diamos. Neural architect: A multi-objective neural architecture search with performance prediction.

[44] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

[45] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. *arXiv preprint arXiv:1707.07012*, 2017.