www.sciencemag.org/cgi/content/full/science.aag2612/DC1

Supplementary Materials for

A generative vision model that trains with high data efficiency and breaks text-based CAPTCHAs

D. George,* W. Lehrach, K. Kansky, M. Lázaro-Gredilla,* C. Laan, B. Marthi, X. Lou, Z. Meng, Y. Liu, H. Wang, A. Lavin, D. S. Phoenix

*Corresponding author. Email: dileep@vicarious.com (D.G.); miguel@vicarious.com (M.L.-G.)

Published 26 October 2017 on *Science* First Release
DOI: 10.1126/science.aag2612

This PDF file includes:

Contents

Organization

This supplementary material is organized in two parts. The first part (Section 1– 7) provides the theoretical foundations of the RCN model and establishes connections with the literature and its biological inspiration. The second part (Section 8) has a more applied focus and provides additional details about RCN's practical implementation, architecture and performance on several benchmark datasets.

In the first, more theoretical part, Sections 1–3 describe the RCN generative model, factorized over shape and appearance. Section 4 describes inference given an input image, and Section 5 describes algorithms for learning the parameters and structure of the model. Section 6 provides context for the present model and establishes connections with the existing literature. We elaborate on the guidance from neuroscience in Section 7.

In the second, more applied part, we describe the details of our preprocessing and post-processing steps in Sections 8.1 and 8.2, respectively. In Section 8.3, a summary of the RCN architectures used throughout the different experiments is given. Experiments on several CAPTCHA datasets, ICDAR, Omniglot, and MNIST (and its noisy variants) are reported in Section 8.4–8.9. The importance of the lateral connections and backward pass is highlighted through a lesion study in Section 8.10. A computational complexity study is performed in Section 8.11 which demonstrates the better scaling behavior of deeper RCN hierarchies. Section 8.12 shows experiments on 3D object renderings. We conclude with some remarks about future work to improve RCN in Section 8.13.

1 Factorizing shape and appearance for object recognition

Human ability to recognize objects is invariant to drastic appearance changes. If we were to see, for example, an entirely blue tree for the first time ever, we would be able to correctly recognize it as a tree and identify its color as blue. Despite the initial surprise, we would not be confused or inclined to think it is a big blueberry. This strongly suggests that we are able to perceive the shape of objects independently of their appearance, and that our categorization of objects relies more strongly on shape cues than on appearance cues. Similarly, even if we never saw an entirely blue tree, we are still able to imagine it by composing our model of "tree" (which contributes a shape) with our idea of "blue" (which contributes an appearance). I.e., our internal representation of objects factorizes shape and appearance and compounds them to form objects.

Based on the above observations, it seems reasonable to expect that an image model with human-level recognition capabilities will have a factorized representation of shape and appearance. Very few works have pursued shape and appearance factorization for image recognition [63, 64]. This factorization enables a model to generalize from fewer examples, since training data only needs to contain images with sufficient diversity of shapes and appearances (and not every combination of them) for the model to come up with a representation that is able to handle the entire cross product space. Despite its success, several mainstream image recognition models, such as the convolutional neural network (CNN) [45, 65], entangle shape and appearance. These models are unable to recognize at test time objects with an appearance that significantly departs from the ones seen *for that particular object* in the training set; they may fail the blue tree test. The obvious solution for these models is to augment their training sets to include images with more combinations of shapes and appearances. Though using trees of every possible color during training would indeed resolve the blue tree problem, using this approach in general results in an unbearable sample complexity.

Our model assumes shape and appearance to be factorized and generates images by combining these two elements using the "coloring book" approach. First the shape of the object, which defines its external and internal boundaries, is generated. Then it is "colored" by having its interior regions filled in with appearance (i.e., with some color or, in general, some texture).

2 A generative hierarchical model for shape

In this section we will describe a probabilistic model that generates an edge map $F^{(1)}$ with the shape of an object. The edge map $F^{(1)}$ will later be combined with the appearance of the object to form the final image X. We call this model the recursive cortical network (RCN). An RCN is a hierarchical latent variable model in which all the latent variables are discrete. We proceed to describe it next.

2.1 Hierarchical model

In order to model the edge map $F^{(1)}$, the RCN uses a hierarchical arrangement alternating *pooling layers* and *feature layers* of latent discrete variables from top to bottom

A very similar arrangement was also used in previous models, such as the convolutional neural network (CNN) [45, 65]. Unlike the CNN, the RCN is a fully generative model, and its properties, even when used for discrimination, are in stark contrast with those of the CNN.

We will use $F^{(\ell)}$ and $H^{(\ell)}$ to collect the latent variables corresponding to the ℓ-th feature layer and pooling layer, respectively. The latent variables in the feature layer are binary, whereas those in the pooling layers are multinomial. The variables of any feature layer $F^{(\ell)}$ can be arranged in a three-dimensional grid, with elements $F^{(\ell)}_{f'r'c'}$. The subscripts refer respectively to *feature* (also called *channel* for clarity), *row*, and *column* of the given layer. Each of the multinomial variables of a pooling layer can also be arranged in a three dimensional grid, with elements $H^{(\ell)}_{frc}$, with the same meaning. There are a total of C layers of each type, numbered from the bottom (closer to the resulting edge map $F^{(1)}$) to the top (closer to the classifier layer $H^{(C)}$), whose role will be detailed below.

The variables of each layer depend only on those of the layer above. Therefore, the joint probability of the model can be written as:

$$\begin{aligned}\log p(F^{(1)}, H^{(1)}, \ldots, F^{(C)}, H^{(C)}) = \log p(F^{(1)}|H^{(1)}) + \log p(H^{(1)}|F^{(2)}) + \\ \ldots + \log p(F^{(C)}|H^{(C)}) + \log p(H^{(C)}).\end{aligned} \tag{S1}$$

All that remains to completely specify the shape model is to describe the probability of a single feature layer (conditional on the pooling layer on top of it) and a single pooling layer (conditional on the feature layer on top of it, if it exists).

2.2 Feature layers

Each of the variables in a feature layer indicates the presence or absence of a given feature f at a given location[1] (r, c) in the image X. For instance, we can have a binary variable at $F^{(3)}$ accounting for the feature "corner at the center of the image". When that variable is turned ON, a slightly distorted corner shape will be generated close to the center of X. The exact shape and location of the corner in X is unknown given $F^{(3)}$; that information is encoded by the layers below it, and would change if we were to fix the values of $F^{(3)}$ and generate multiple samples of X.

[1]For simplicity, we will assume in this manuscript that no subsampling is occurring in the layers of the RCN (i.e., in CNN terminology, that we are using a stride of 1). This produces a one-to-one correspondence between the (r, c) locations in any feature layer $F^{(\ell)}$ or pooling layer $H^{(\ell)}$ and the image X, which are useful for description purposes. Subsampling is indeed possible (and computationally desirable) in this architecture, and is analogous to using stride values above 1 in CNNs.

Each individual feature variable is independent of the others given the layer above, so we can write

$$\log p(F^{(\ell)}|H^{(\ell)}) = \sum_{f'r'c'} \log p(F^{(\ell)}_{f'r'c'}|H^{(\ell)}). \tag{S2}$$

Usually, $F^{(\ell)}_{f'r'c'}$ will only depend on a small subset of the variables $H^{(\ell)}$. Each pool variable in $H^{(\ell)}$ is multinomial and has several states, each of them associated to one of the variables in $F^{(\ell)}$, plus a special state which is called the OFF state. The set of feature variables to which a pool is associated are called the *pool members*. Multiple pools can share the same pool member.

To make the above idea more concrete, we will consider a type of connectivity from pools to features called "translational pooling". In translational pooling, each pool $H^{(\ell)}_{frc}$ has as pool members the elements $\{F^{(\ell)}_{f'r'c'} : |r' - r| < \text{vps}^\ell, |c' - c| < \text{hps}^\ell, f' \text{ constant}\}$, where vps^ℓ and hps^ℓ are the vertical and horizontal pool shapes respectively, and f' is the feature that $H^{(\ell)}_{frc}$ endows with local translation ability. In the common case in which square pools are used, we use a single parameter to define their height and width: pool size $= 2 \times \text{hps}^\ell - 1 = 2 \times \text{vps}^\ell - 1$. A pool can either be OFF or activating any of the feature variables with feature index f' and in the vicinity of its position (r, c). Note that a pool can only activate a single pool member at a time.

Once the connectivity between a pooling layer and the feature layer immediately below has been set (for instance, using translational pooling) the conditional probability of each feature variable is simply

$$p(F^{(\ell)}_{f'r'c'} = 1|H^{(\ell)}) = \begin{cases} 1 \text{ if any pool in } H^{(\ell)} \text{ is in a state associated to } F^{(\ell)}_{f'r'c'} \\ 0 \text{ otherwise} \end{cases} . \tag{S3}$$

In other words, $F^{(\ell)}$ is deterministic given $H^{(\ell)}$. There are multiple $H^{(\ell)}$ that can result in the same $F^{(\ell)}$, so the relation is not bijective.

The top feature layer $F^{(C)}$ is assumed to contain complete objects. At $F^{(C)}$ each channel accounts for a different object type, whereas each (r, c) location within that channel accounts for a different location of the generated object.

2.3 Pooling layers

Unlike feature layers, pooling layers[2] are not deterministic, so all the variability in image generation from an RCN comes from the pooling layers. This makes them quite a bit more complex than feature layers.

As explained before, each multinomial pool variable in a pooling layer $H^{(\ell)}$ can be in any of multiple states, one of which is the OFF state and the others are feature variables of layer $F^{(\ell)}$, which form the members of that pool. One can think of the members of a pool as different alternatives that express slight variations of the same feature. For instance, the pool members of a translational pool will be features that look identical but are located at different positions, within a radius of the pool position.

There is an implicit top level pooling layer $H^{(C)}$ over classifier features. This layer has a single multinomial variable that chooses among the different top-level features.

[2]Since we are taking the generative perspective, readers familiar with the CNN literature might prefer to think of them as *unpooling layers*. Since this is a probabilistic model, the same layer can actually be performing pooling or unpooling operations depending on the task (sampling or inference).

The performance of RCN can be significantly improved by the use of *lateral connections*. We will first consider the simpler no-laterals case and then move on to the laterally connected model.

2.3.1 No lateral connections

In the absence of laterals, each individual pool state is also independent of all others given its parent on the layer above, so we can write

$$\log p(H^{(\ell)}|F^{(\ell+1)}) = \sum_{frc} \log p(H^{(\ell)}_{frc}|F^{(\ell+1)}) = \sum_{frc} \log p(H^{(\ell)}_{frc}|F^{(\ell+1)}_{f'r'c'}), \tag{S4}$$

where $F^{(\ell+1)}_{f'r'c'}$ is the feature that activates pool $H^{(\ell)}_{frc}$ (i.e., $F^{(\ell+1)}_{f'r'c'}$ is the parent of $H^{(\ell)}_{frc}$). Note that each pool variable has a single parent (but each feature variable can have multiple children).

Each feature $F^{(\ell+1)}_{f'r'c'}$ activates some of the pools located close to it; i.e., some of the pools $\{H^{(\ell)}_{frc} : |r'-r| < \text{vfs}^\ell, |c'-c| < \text{hfs}^\ell\}$, where vfs^ℓ and hfs^ℓ are respectively the vertical and horizontal feature shape. Note that a feature variable at level $\ell+1$ and channel f' can activate a set of pools at layer ℓ at multiple *different* channels $\{f\}$, whereas in the previous section we saw that, given translational pooling, a pool at level ℓ and channel f can only activate features at level ℓ with all of them residing in the *same* channel f'. Another important distinction with the previous section is that two different features cannot activate the same pool (unlike pools, which can have the same feature variable within their pool members, sharing it). In the cases in which it would be desirable for multiple features to activate the same pool, we can just make a copy (i.e., create an additional channel in that pooling layer with the same connectivity to the layers below) and use that pool instead.

A feature can be described in terms of the position (relative to its own) of the pools that it activates. Whenever any of the features that activate a given pool $H^{(\ell)}_{frc}$ is set to 1, then the pool cannot be in the OFF state. This defines the probability density of the pooling layers as

$$p(H^{(\ell)}_{frc} = \text{pool member}_m|F^{(\ell+1)}_{f'r'c'}) = \begin{cases} 1/M \text{ if parent feature is 1} \\ 0 \text{ if parent feature is 0} \end{cases}, \tag{S5}$$

where M is the number of pool members for that pool. This of course implies that $p(H^{(\ell)}_{frc} = \text{OFF}|F^{(\ell+1)}_{f'r'c'}) = 1$ when the parent feature is 0.

This in turn means that the joint probability of the latent variables for a valid assignment (one that does not result in the joint probability being zero) is

$$\log p(F^{(1)}, H^{(1)}, \ldots, F^{(C)}, H^{(C)}) = \sum_{\text{active pool}_j} \log \frac{1}{\#\text{ pool members of active pool}_j}. \tag{S6}$$

The top layer $H^{(C)}$ in Eq. S1 is a pool with no parents. The OFF probability for that pool is 0 (it always activates exactly one pool member), and contains as pool members all the feature variables of the next layer. That means it can generate any object (we regard the top-level features as entire objects) at any location. This pooling layer is special in the sense that it only has a single pool, whereas all the other pooling layers, even if they consist of a single channel, have multiple pools following a topographic arrangement.

The hierarchical model as described so far is directly useful to model edge maps, but turns out to be too flexible in many cases. If we use translational pools, we can control the amount of distortion in the generation process by changing the pool shape. But we found that no setting of this value produced entirely reasonable results: small values resulted in too rigid images that could not adapt

to the distortions found in actual images (e.g., a corner made of many pools with close-to-zero pool shape results in two almost rigid edges), while large values resulted in shapes with discontinuous edges (e.g., the previous corner would look like a cloud of points with no clear edges, since each pool choice is independent of the rest).

To solve the above problem, we would like a feature to spawn multiple pools that behave in a coordinated way – a corner shape could be distorted while still looking like a corner and exhibiting edge continuity.

2.3.2 Lateral connections

In the no-laterals case, the term $p(H^{(\ell)}|F^{(\ell+1)})$ is fully factorized, as shown in Eq. S4. This implies no coordination among the pool choices. We can coordinate these pool choices by entangling all the pool choices belonging to the same parent. The factorization is now

$$\log p(H^{(\ell)}|F^{(\ell+1)}) = \sum_{f'r'c'} \log p(\{H^{(\ell)}_{frc} : \text{ whose parent is } F^{(\ell+1)}_{f'r'c'}\}|F^{(\ell+1)}_{f'r'c'}). \tag{S7}$$

We can then define the joint the probability of a set of pools with a common parent by introducing pairwise constraints between some (or all) of the pool states. Consider a set of pools $\{H_j\}$ with a common parent feature F, where we drop layer and position indicators for simplicity. According to the previous description, F is defined by a set of triplets $(f_1, \Delta r_1, \Delta c_1), \ldots, (f_J, \Delta r_J, \Delta c_J)$ specifying which pools in the layer below it should activate, with their positions being encoded relative to the position of F itself. We can augment that information by including a set of pairwise constraints between the states of pools i and j, which we will denote s_i and s_j, $\{c_{Fij}(s_i, s_j)\}$ that indicates which pairs of states are allowed (those for which the constraint has value 1) and which are not.

With each feature F specifying its own set of constraints $\{c_{Fij}(s_i, s_j)\}$ for its children pools, we get the following joint density:

$$p(H_1 = s_1, H_2 = s_2, \ldots, H_J = s_J|F = 1) = \begin{cases} 1/S_F \text{ if } \forall_{ij} c_{Fij}(s_i, s_j) = 1 \\ 0 \text{ otherwise} \end{cases}, \tag{S8}$$

where S_F is the number of joint states allowed by the constraints of F. This of course implies

$$p(H_1 = \text{OFF}, H_2 = \text{OFF}, \ldots, H_J = \text{OFF}|F = 0) = 1. \tag{S9}$$

In this case, the joint probability of the latent variables for a valid assignment is

$$\log p(F^{(1)}, H^{(1)}, \ldots, F^{(C)}, H^{(C)}) = \sum_{F_j : F_j = 1} -\log(S_{F_j}). \tag{S10}$$

There are many sets of pairwise constraints that could produce satisfactory pool coordination effects. A typical pool coordination constraint in our work are called *perturb laterals*: we first expand each state s_j of pool j (which is located at (r_j, c_j)) to the tuple describing the member feature to which it is associated $s_j \equiv (\Delta r_j, \Delta c_j)$, with the delta positions being relative to the pool center, and the channel f of each member feature being the same as that of its pool. Then, the perturb lateral constraint can be written as follows:

$$c_{ij}(s_i, s_j) = c_{ij}((\Delta r_i, \Delta c_i), (\Delta r_j, \Delta c_j)) = |\Delta r_i - \Delta r_j| < |r_i - r_j|/\text{pf}^\ell \wedge |\Delta c_i - \Delta c_j| < |c_i - c_j|/\text{pf}^\ell, \tag{S11}$$

where pf^ℓ is the perturbation factor for layer ℓ. In this way, the maximum perturbation in the relative position of two pools is proportional to their distance, with pf^ℓ being the (inverse) proportionality constant.

2.4 Translational invariance

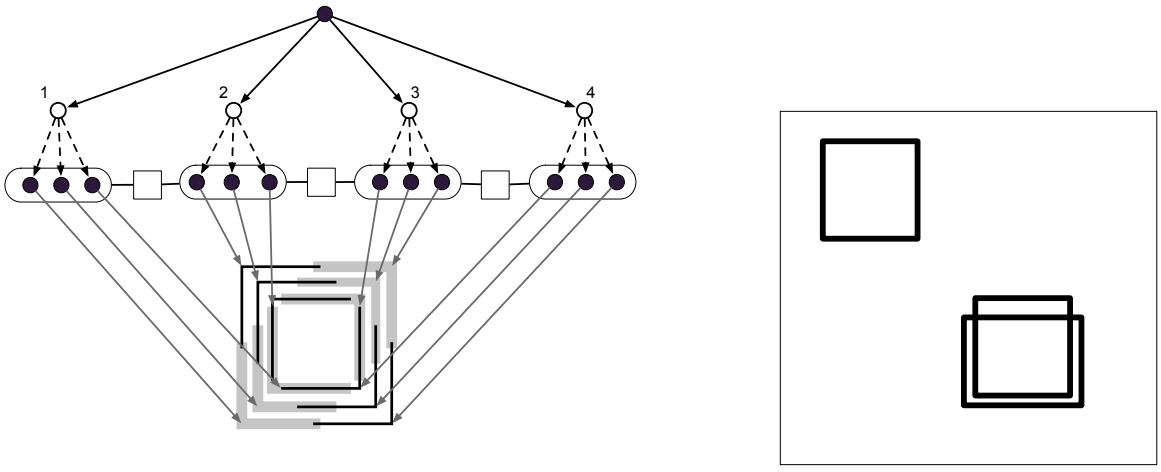
Let us sum up the parameters of this architecture:

- Feature parameters, $(f_1, \Delta r_1, \Delta c_1), \ldots, (f_J, \Delta r_J, \Delta c_J)$ and $\{c_{Fij}(s_i, s_j)\}$, specifying connectivity to pools in the layer below and constraints among their selected pool members when the feature is active. These parameters are potentially different for each binary feature variable in the entire hierarchy.
- Pool parameters, $(\Delta r_1, \Delta c_1), \ldots, (\Delta r_M, \Delta c_M)$, specifying connectivity to pool members in the feature layer below. These parameters are potentially different for each pool in the hierarchy.

Even though this is not necessarily a property of the above hierarchy, in vision it is generally useful to have a model that shows equivariance under translations of the input. Equivariance in this case means that, given an image and a set of hidden variables, the model's likelihood will not change if we shift the input image *and the position of the values of all the hidden variables* by the same amount. The hidden variable $H^{(C)}$ is unique and cannot be shifted, so instead its value will be shifted accordingly.

This can easily be achieved in our hierarchy if we set the parameters above corresponding to the same channel and layer to be identical. This is the convolutional assumption from which CNNs are named. Additionally, this results in a reduction of the free parameters in the model.

Because child features are allowed to overlap, local translations of features can also produce size invariance of a higher-level feature or object, as shown in Fig. S1.



(a) Size variations (b) Input with multiple rectangles

Figure S1: (a) Representation of a rectangle as a conjunction of four pools, each representing a corner. Translations of the corners can represent size variations of the rectangle because RCN allows child features to overlap. The corner features are rendered in different thicknesses and shades to show the overlap between them. Each corner feature itself can be broken down further into line segment features to produce more local variations. Note that different aspect ratios and translations of a rectangle can be generated from the same higher-level feature node. The factor between pool 1 and pool 4 is omitted for clarity. (b) An input image with multiple rectangles. When the rectangles are well separated, they will be represented by different top-level feature nodes at different translations. Since each of the overlapping rectangles could be represented by a single top-level feature node, explaining both the rectangles would require two copies of the same feature at the top level. In general, it can be assumed that the same feature node is copied a fixed number of times, allowing for multiple instances of the same object to appear roughly at the same location. In practice, this effect can also be approximately achieved without making copies of the features by using the same hierarchy multiple times until all evidence is explained. See section 4.7 for more details on scene parsing.

3 Combining shape and appearance

In this section, we show how to combine the shape model described in the previous section with an appearance model. The previous model generates the edge map $F^{(1)}$, which is an array of binary variables indexed by (f, r, c). Each of the features f corresponds to a *patch descriptor*, which is defined by a set of IN and OUT variables and a set of edges arranged inside a rectangular patch (see Fig. S2).

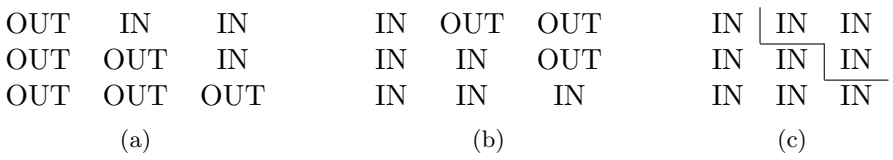


Figure S2: Patch descriptors: the three possible orientations of a 45° edge and 3×3 *patch size*.

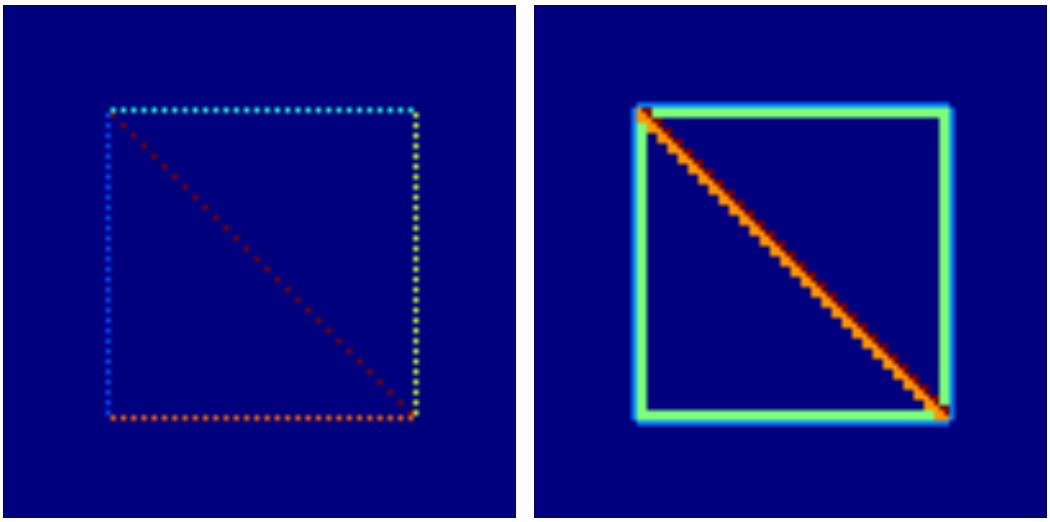
When a variable at (f, r, c) within the edge map $F^{(1)}$ is turned on, a small edge of orientation f will appear in the final image at the row and column (r, c). The edges are oriented because, in the case of an outer edge, one of the sides is defined to be IN while the other is defined to be OUT. For instance, there are two separate features corresponding to an outer vertical edge. One of them defines IN to be on the left of the edge (and OUT on the right) whereas the other feature defines IN to be on the right of the edge (and OUT on the left). In the case of an inner edge in an object (i.e., an edge that separates two interior regions of the object), both sides of the edge are defined as IN. For inner edges there is additional information showing the border between the IN values of each side of the edge, as depicted in Fig. S2(c). Fig. S3(a) contains two types of vertical outer edges, two types of horizontal outer edges, and a diagonal inner edge (which is precisely the one shown in Fig. S2(c)). Therefore, for each rotation of an edge, there are three possible orientations: two corresponding to outer edges plus one corresponding to an inner edge. A typical number of edge map features is 48, accounting for the three orientations of 16 different rotations.

Let us denote by Y the canvas on which both the oriented edges of the edge map are going to be drawn, and the interiors of objects are going to be filled in with some texture or color. Y is a 2D array of multinomial random variables, indexed by row and column (r, c). The row and column indices correspond to the row and column of pixels in the final image; there is a one-to-one correspondence between the variables in Y and the observed image pixels X. The state of each multinomial variable of Y represents both a color (or texture) and the IN or OUT state. Therefore, each variable of Y has twice as many states as required to represent all possible colors (or textures).

Y is a conditional random field (CRF) with a fixed structure of 4-connected variables, so that each variable has an edge connecting it to the 4 other variables in its immediate neighborhood. These edges represent pairwise potentials, and there also exist per-variable unary potentials. The potentials depend on the values of $F^{(1)}$ (making the random field conditional). Each active variable in $F^{(1)}$, following its patch descriptor (see Figs. S2 and S3), forces some of the variables of Y to be in an IN or an OUT state, and for inner edges modifies the pairwise potentials between variables in an IN state that belong to different regions, as depicted in Fig. S2(c). The set of pairs of adjacent locations in Y with an inner edge active between them (according to $F^{(1)}$) is called IE. The remaining set of pairs of adjacent locations (i.e., all those that do not have an inner edge activation between them coming from $F^{(1)}$), are called $\overline{\text{IE}}$.

The variables of Y obey the following conditional density:

$$p(Y|F^{(1)}) = \frac{1}{Z_{F^{(1)}}} \prod_{(r,c)} \Phi(Y_{rc}) \prod_{(r,c),(r',c')\in \text{IE}} \Phi_{\text{IE}}(Y_{rc}, Y_{r'c'}) \prod_{(r,c),(r',c')\in \overline{\text{IE}}} \Phi_M(Y_{rc}, Y_{r'c'}) \Phi_E(Y_{rc}, Y_{r'c'}). \tag{S12}$$



(a) Visualization of $F^{(1)}$ (b) Influence of $F^{(1)}$ on Y

Figure S3: (a) Visualization of the 3D binary variables $F^{(1)}$. Dark blue represents zero for all channels at the given (r, c). Other colors stand for a one at a given channel of (r, c), while the remaining channels are set at zero (of course, multiple channels could activate at the same (r, c), but not in this specific example so it is easier to visualize). The red feature (in the diagonal), corresponds to the patch descriptor in Fig. S2(c). (b) The joint distribution of Y is conditioned on $F^{(1)}$: Light blue pixels are forced to be in an OUT state, while green, orange, and red pixels are forced to be in a state of type IN. The factors between orange and red variables are included in the set IE, whereas all the other factors are included in the set $\overline{\text{IE}}$.

To clarify the above description with an example, consider the effect that setting $F^{(1)}_{4,2,2} = 1$ and the remaining variables of $F^{(1)}$ to 0 causes on Y. Imagine that the patch descriptor of feature 4 is the one shown in Fig. S2(c), where (1,1) indexes the top-left corner. Then, following the continuous line boundary drawn in Fig. S2(c), the pairs of variables (Y_{11}, Y_{12}), (Y_{22}, Y_{12}), (Y_{22}, Y_{23}), and (Y_{33}, Y_{23}) will have an inner edge between them and therefore be connected by pairwise potentials of the form $\Phi_{\text{IE}}(Y_{rc}, Y_{r'c'})$. Every other pair of adjacent variables in Y will be connected by pairwise potentials of the type $\Phi_M(Y_{rc}, Y_{r'c'})\Phi_E(Y_{rc}, Y_{r'c'})$. The pairwise potentials are designed to favor the continuity of the IN and OUT regions, as well as the appearance of edges in transitions between IN and OUT regions, and between IN of different regions.

In particular, the pairwise potential between two adjacent variables that $F^{(1)}$ indicates as belonging to different inner regions of the object is

$$\Phi_{\text{IE}}(Y_{rc}, Y_{r'c'}) = \begin{cases} 1 \text{ if } d(Y_{rc}, Y_{r'c'}) > \alpha \\ \beta \text{ otherwise (with } 0 < \beta < 1) \end{cases}, \tag{S13}$$

whereas the remaining pairwise potentials between adjacent variables are

$$\Phi_M(Y_{rc}, Y_{r'c'}) = \begin{cases} 1 \text{ if } Y_{rc} \text{ and } Y_{r'c'} \text{ are both of type IN or both of type OUT} \\ \gamma \text{ otherwise (with } 0 < \gamma < 1) \end{cases}, \tag{S14}$$

(i.e., a Potts model favoring contiguous IN and OUT regions) and

$$\Phi_E(Y_{rc}, Y_{r'c'}) = \begin{cases} 1 \text{ if } Y_{rc} \text{ and } Y_{r'c'} \text{ are of type OUT} \\ 1 \text{ if } Y_{rc} \text{ and } Y_{r'c'} \text{ are of type IN and } d(Y_{rc}, Y_{r'c'}) \leq \alpha \\ 1 \text{ if } Y_{rc} \text{ and } Y_{r'c'} \text{ are of different type and } d(Y_{rc}, Y_{r'c'}) > \alpha \\ \beta \text{ otherwise (with } 0 < \beta < 1) \end{cases}, \tag{S15}$$

(i.e., a model generating outer edges). Finally, the unary potentials that connect the CRF to the shape model are

$$\Phi(Y_{rc}) = \begin{cases} 0 \text{ if } Y_{rc} \text{ is of type IN (OUT) but should be of type OUT (IN) according to } F^{(1)} \\ 1 \text{ otherwise} \end{cases}, \tag{S16}$$

The distance $d(Y_{rc}, Y_{r'c'})$ between the values of two canvas variables is a color or texture distance (depending on the whether Y_{rc} is taken to represent colors or textures). The parameters α and β specify the maximum acceptable distance between two adjacent pixels and the penalization for overstepping it, respectively. The distance between two variables ignores the type of the variable being IN or OUT. Transitions between these two types of states are penalized by γ. Since multiple variables (up to nine in our running example using a 3×3 patch size) from $F^{(1)}$ can potentially determine the IN-or-OUT type of the same variable of Y, when the case of conflicting determinations over the same variable arises, no determination is made for that variable. The constant $Z_{F^{(1)}}$ ensures proper normalization of the probability and we emphasize that its value will depend on the values of $F^{(1)}$.

This makes $p(Y|F^{(1)})$ essentially a Potts model plus an edge generation model. The values of some variables are clamped to be of type IN or OUT and some of the pairwise potentials are modified to favor high contrast values on each region of an inner edge. The qualitative behavior of this model is easy to understand: $F^{(1)}$ clamps where the edges of the object should be located, whereas the entanglement among neighbors of the CRF fills in the interiors of the object with an arbitrary but consistent color or texture.[3]

We will assume that $F^{(1)}$ defines watertight regions in Y, as shown in Fig. S3(b). This is approximately true for any reasonable shape model. The conditioning process that defines the CRF must fill in any gaps to make the regions exactly watertight. We can additionally assume that $\gamma \to 0$, so that the IN and OUT types of the variables will propagate inside each watertight region, without any flips.

Note that the maximum probability configuration of the CRF corresponds to filling in each watertight region with slowly varying states (change rate below α per pixel) and of type IN, while keeping contrast across borders (state jump above α). When $\beta = 0$, a solid color fills each region in the MAP configuration, with contiguous regions showing the required α contrast.

While sampling from the shape hierarchy is trivial, sampling from this CRF is not as easy. We used Gibbs sampling to draw eight independent samples from Y when conditioning on the $F^{(1)}$ of Fig. S3(a), displayed in Fig. S4. Each sample was obtained with 1000 Gibbs updates of each random variable.

Finally, the pixels of the observed image (i.e., the elements of X) can be probabilistically defined in terms of their local elements[4] within variable Y. For the case in which the states of Y represent colors[5], we can consider that each pixel in X is only influenced by the corresponding element of Y. This would be the extremest case of locality and results in the joint conditional factorizing over pixels, $\log p(X|Y) = \sum_{r,c} \log p(X_{rc}|Y_{rc})$. The type IN or OUT of Y_{rc} is irrelevant in the computation of the conditional. In this case, we can say that each of the elements of X is defined as the output of a noisy channel that takes the corresponding element of Y as input.

Unless stated otherwise, we will use as a default noisy channel $p(X_{rc}|Y_{rc})$ a mixture of two uniform densities: the first assigns constant probability to all possible states of X_{rc}, whereas the second assigns

[3]For now we assume a fixed set of exemplars in color/texture space. This set could be learned from unlabeled data in a separate step.

[4]For a given element within X, X_{rc}, the local elements of Y are those that are close to it in the spatial arrangement – i.e., $\{Y_{ab} : |a - r| < d, |b - c| < d\}$, for some radius d.

[5]One could also use this extreme level of locality in the case in which Y represents textures, but the richness of the textures would be severely limited.

constant probability to the states of X_{rc} within some radius of the state of Y_{rc} (since states represent colors, the distance is measured in color space). This noisy channel has two parameters: one specifying the mixture weight and the another specifying the radius. For any valid noisy channel (including the default just described), we will use $\text{argmax}_{Y_{rc}}\, p(X_{rc}|Y_{rc}) = X_{rc}$, even though other values of Y_{rc} could also maximize the previous expression.

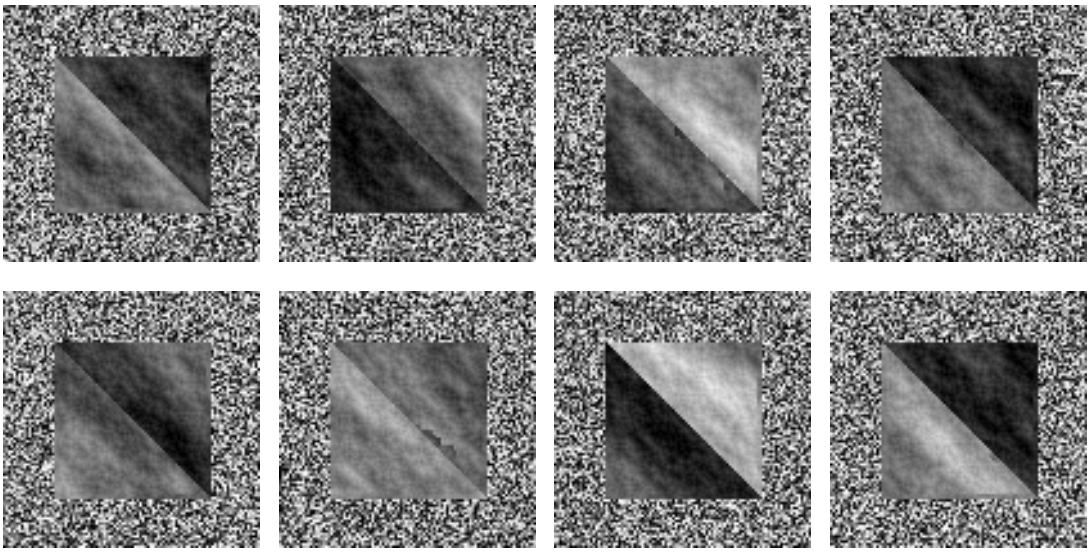


Figure S4: Eight different random samples of Y drawn from $p(Y|F^{(1)})$ when $F^{(1)}$ is fixed to the value of Fig. S3(a). Each variable has 64 states, 32 of type IN and 32 of type OUT. Each of the 32 appearances has been given the interpretation of a different shade of gray to form the resulting image. The parameters were set to $\alpha = 2$, and $\beta, \gamma \to 0$.

4 Inference

The above model generates images corresponding to different isolated objects. Once the parameters of the hierarchical model have been set using some training collection of isolated objects, recognition amounts to inference in the probabilistic model. In particular, recognizing an isolated object amounts to inferring the value of the top pool $H^{(C)}$, which informs us about both the category and location of the object.

When background clutter or multiple objects are present in the same image, we can still use the above procedure inside an outer loop to find the most salient objects in the image, sequentially masking them out and looking for the next most salient objects.

Exact inference on such a complex, loopy model is of course intractable. In this section we will show how approximate inference can be performed in this model. We will show that a single bottom-up *forward pass* will be enough to find a good approximation of the location and categories of the objects, and that a subsequent single top-down *backward pass* will be enough to find the masks corresponding to each of the recognized objects.

Ideally, we would like to compute the full posterior over the latent variables of the model. However, this is intractable. Approximate inference algorithms on loopy graphs such as loopy belief propagation [38] and more advanced variants [66] can be used to approximate the marginal posterior probability of each latent variable. These techniques are not guaranteed to compute the actual marginal posterior probabilities, but even if they were, they are not appropriate to address the problem at hand. The reason is that approximating the true posterior as the product of the marginal posterior probabilities introduces non-zero posterior probability mass in regions in which the actual posterior probability

might be zero[6]. In other words, attempting to compute the marginal posteriors of the latent variables would result in locally valid probabilities, but would not exhibit global coordination and consistency. Instead, finding a single configuration of the latent variables that maximizes the probability of the observed image will result in a consistent, globally coordinated latent explanation of the observed image. Therefore, we will be interested in maximum a posteriori inference (MAP).

MAP inference in loopy graphs can be approximately solved using techniques such as max-product loopy belief propagation [38], tree-reweighted belief propagation [67] dual decomposition [68], etc. Our overarching approximate inference procedure will consist of a single forward and backward pass using max-product message passing. Some of the messages can be readily computed, whereas others will require some sort of approximation. In particular, to solve the subproblems involving lateral connections (the *lateral subproblem*), we will either use loopy max-product locally or more advanced dual decomposition techniques[7].

4.1 Loopy belief propagation: max-product

Max-product belief propagation was developed as an exact technique to find the states of the variables that maximize the probability of a tree-structured graphical model [38]. However, it was empirically shown that when max-product message passing was applied to loopy graphs, it achieved competitive performance.

Any probabilistic model (such as the RCN model) can be expressed as a factor graph, where the factors that entangle groups of variables are made explicit

$$p(x) = p(x_1, x_2, \ldots, x_n) = \prod_c \phi_c(x_c), \tag{S17}$$

where c runs over the factors in the graph, and x_c are subsets of some set of variables $x_1, x_2, \ldots, x_n$ in the graph[8]. MAP inference aims at finding $x^* = \text{argmax}_x\, p(x_1, x_2, \ldots, x_n)$. If we call $x_{c\setminus i}$ to the variables in x_c excluding variable x_i, we can define a "message" from each factor c to each of its variables i as $\hat{m}_{c\to i}(x_i)$ and the max-product message updating rule is

$$\hat{m}_{c\to i}(x_i)^{\text{new}} = \max_{x_{c\setminus i}} \left[\log \phi_c(x_i, x_{c\setminus i}) + \sum_{j\in c\setminus i} \left(\mu(x_j) - \hat{m}_{c\to j}(x_j)\right) \right], \tag{S18}$$

where $\mu(x_i)$ is the estimation of the max-marginals for variable x_i (expressed, like the messages, in the log-domain):

$$\mu(x_i) = \sum_{c: i\in c} \hat{m}_{c\to i}(x_i). \tag{S19}$$

An approximate solution to the MAP problem is obtained by setting $x_i = \text{argmax}_{x_i}\, \mu(x_i)$ after running loopy max-product. Message updating can be run until convergence (which is not guaranteed

[6]For instance, if two binary variables x and y have prior probability $p(x = 1) = p(y = 1) = 0.5$ and we observe its sum to be 1, then the exact posterior marginals on each of the variables is $p(x|x + y = 1) = p(y|x + y = 1) = 0.5$. Using this as a factorized posterior approximation places a significant posterior probability of 0.25 on the *impossible* configuration $x = 1$, $y = 1$.

[7]We observed that dual decomposition improves the quality of the solution, but the difference is not drastic. When the size of the lateral graph involves many not-so-sparsely connected pools, which tends to be the case when using the two-level version of the hierarchy, the speed benefit of loopy max-product without substantial performance loss can be preferable.

[8]This is a completely general description of max-product inference, variables x might refer to any of the variables in our model.

in loopy graphs) or for a fixed number of iterations. The algorithm is invariant to message renormalization, which in the log-domain means that we can add an arbitrary constant to each message without changing the resulting solution. The discrete variables in our model have a default state (OFF for pools, 0 for features), so during propagation we will use the following normalized messages instead

$$m_{c\to i}(x_i) = \hat{m}_{c\to i}(x_i) - \hat{m}_{c\to i}(x_i = \text{default value}), \tag{S20}$$

which allows for a more compact message encoding. For instance in the case of binary variables, the single scalar $m_{c\to i}(x_i = 1)$ will be needed (since it is known that $m_{c\to i}(x_i = 0) = 0$).

4.2 Forward pass

In this pass, we start from the observed image and progress towards the top pool of the hierarchy $H^{(C)}$ performing max-product updates in our way up. A variable can be connected to multiple factors. The aggregated messages coming from all factors in the layer below are called the "bottom-up message" to that variable. Analogously, the "top-down message" aggregates the messages coming from factors connecting to variables in the layer above. Our schedule starts with a forward pass that updates one layer at a time from the bottom to the top, and updates all the bottom-up messages to the variables of each layer in parallel. The top-down messages (in log space) are initialized to $-\infty$.

We have experimented with different variations of the forward pass and we have found that in most scenarios the increased cost of considering the lateral connections does not result in a significant improvement in the final solution, so we will often run our forward pass as if the lateral connections did not exist.

The details of the updates for each of type of layer are explained next.

4.2.1 The preprocessing layer

The preprocessing layer interfaces the observed natural image X with the discrete layer $F^{(1)}$, approximating the effect of the canvas layer Y.

The output of the preprocessing layer are the bottom-up messages that are fed as bottom-up input to $F^{(1)}$. According to the max-product rule described in Section 4.1 and the initialization of the top-down messages, the bottom-up messages should be

$$m_{\text{bu}}(F^{(1)}_{frc}) = \max_Y \log p(X, Y | F^{(1)}_{frc} = 1, \{F^{(1)} \backslash F^{(1)}_{frc}\} = 0) - \max_Y \log p(X, Y | F^{(1)} = 0), \tag{S21}$$

where $\{F^{(1)} \backslash F^{(1)}_{frc}\}$ are all the feature variables in layer 1 other than $F^{(1)}_{frc}$. This corresponds to the likelihood ratio of edge $F^{(1)}_{frc}$ being present in canvas Y and image X when all the other edges are known to be off. We could indeed use message propagation through the (very loopy) canvas CRF Y and then propagate up to $F^{(1)}$. However, this would be a costly method to detect the presence of an oriented edge in an image. Instead, we consider two alternatives to estimate $m_{\text{bu}}(F^{(1)}_{frc})$.

Shape and appearance preprocessing layer Consider for the sake of simplicity and concreteness that the states of the canvas Y (see Section 3) correspond to colors, each of which can additionally have IN or OUT type.

In this case, it is easy to see that $\max_Y \log p(X, Y | F^{(1)} = 0) = -\log Z_{F^{(1)}=0} + N_{\text{pixels}} \log m_n$, where $m_n = \max_{X_{rc}, Y_{rc}} p(X_{rc} | Y_{rc})$ is the maximum probability that an individual pixel can attain under the noisy channel. First, note that the factors that compose $p(X, Y | F^{(1)} = 0)$ are $p(X|Y)p(Y|F^{(1)} = 0)$.

Since in this case the edge map is not forcing any edges or restricting any variable within Y to be of type IN, the configuration of Y that maximizes $p(X|Y)p(Y|F^{(1)}=0)$ corresponds to setting all variables to type OUT and the color that matches X. The color matching between X and Y sets $\log p(X|Y)$ to its maximum value[9] $N_{\text{pixels}} \log m_n$ and having all variables within Y set to an OUT-type state sets $\log p(Y|F^{(1)}=0) = -\log Z_{F^{(1)}=0}$ (all the log-factors take value zero), also its maximum value. Therefore the maximum value of $\log p(X,Y|F^{(1)}=0)$ is just $N_{\text{pixels}} \log m_n$ minus its log-partition function.

Finding the configuration of Y that maximizes $\log p(X,Y|F^{(1)}_{frc}=1, \{F^{(1)} \backslash F^{(1)}_{frc}\}=0)$ in general is not easy. Assume that we are activating an oriented edge with the patch descriptor of Fig. S2(a). A valid (but not necessarily optimal) configuration Y^* results if we set the color of each variable in Y to match the color of X and its type to be OUT except for those that are forced to be IN by the active edge. For this particular choice of Y, we have

$$\begin{aligned}\log p(X,Y|F^{(1)}_{frc}=1, \{F^{(1)} \backslash F^{(1)}_{frc}\}=0) &= \log p(X|Y^*) + \log p(Y^*|F^{(1)}_{frc}=1, \{F^{(1)} \backslash F^{(1)}_{frc}\}=0) \\ &= N_{\text{pixels}} \log m_n + v_E \log \beta + t_M \log \gamma - \log Z_{F^{(1)}_{frc}=1, \{F^{(1)} \backslash F^{(1)}_{frc}\}=0},\end{aligned} \tag{S22}$$

where t_M and v_E are the number of mask transitions and *edge violations* respectively. A mask transition occurs whenever there is a transition from IN to OUT (or vice versa) anywhere in the CRF. An edge violation occurs when the distance between colors inside the IN region is larger than α, or when the distance between colors in a transition from IN to OUT (or to the IN of a different region if the active edge was an inner edge) is smaller than α. These violations are trivial to check and are local[10] to the edge patch under consideration, so they can be computed efficiently. Of course, using Y^* results only in approximate maximization. This results in the approximate bottom-up message

$$m_{\text{bu}}(F^{(1)}_{frc}) = v_E \log \beta + t_M \log \gamma - \log Z_{F^{(1)}_{frc}=1, \{F^{(1)} \backslash F^{(1)}_{frc}\}=0} + \log Z_{F^{(1)}=0} = v_E \log \beta + b_p, \tag{S23}$$

where computing the difference of log partition functions is not tractable, but does not depend on the observed image and together with $t_M \log \gamma$ will be approximately constant[11], so can be considered as a parameter of the model, b_p. Observe the functional rôle of the preprocessing layer: it is an interface between the natural image and the discrete shape model, it is detecting regions of the input image in which a given oriented edge is likely to exist and additionally checking for appearance (color, texture) consistency within the region(s) that are considered to be inside the object. The degree to which these two desires are met can be quantified in terms of the number of violations with respect to the edge patch descriptor. In practice, any function that detects oriented edges and appearance consistency can be used at this preprocessing stage.

Shape only preprocessing layer An even simpler option is to use a preprocessing stage that discards the appearance consistency information. Such preprocessing stage only needs to perform edge detection at multiple rotations (without considering for each rotation the three different orientations described in Fig. S2) and produces only a small performance degradation in practice. Any edge detection algorithm such as Gabor filtering can produce satisfactory results.

[9] See the last paragraph of Section 3.

[10] Since every variable outside the patch is set to the OUT type, all violations must occur inside the edge patch or its limits.

[11] The number of mask transitions might be slightly different for some rotations due to the effect of discretization. This effect is less noticeable as the patch size of the descriptors grows. Similarly, due to border effects, the partition function $\log Z_{F^{(1)}_{frc}=1, \{F^{(1)} \backslash F^{(1)}_{frc}\}=0}$ might have a slight dependence on r, c. These effects are small and ignored.

4.2.2 Feature layers

During the forward pass, all the binary variables in feature layer $F^{(\ell)}$ receive bottom-up messages $m_{\text{bu}}(F^{(\ell)}_{f'r'c'})$, which are a scalar value. Those can come from the preprocessing layer (for $\ell = 1$) or from a pooling layer (for $\ell \neq 1$). As described in Section 2.2, each variable $F^{(\ell)}_{f'r'c'}$ can be activated by a single state of some of the pools at $H^{(\ell)}$. Conversely, a given pool $H^{(\ell)}_{frc}$ activates a different feature with each of its states. Consider the set of features that $H^{(\ell)}_{frc}$ can activate to be called $\{F^{(\ell)}_i\}$. In particular, if we named the states such that feature $F^{(\ell)}_i$ is activated whenever pool $H^{(\ell)}_{frc}$ is in state i, the max-product rule tells us that the bottom-up message to $H^{(\ell)}_{frc}$ is simply:

$$m_{\text{bu}}(H^{(\ell)}_{frc} = i) = m_{\text{bu}}(F^{(\ell)}_i = 1). \tag{S24}$$

This means that the forward pass through feature layers simply routes the bottom-up messages to all[12] the pool states that activate them.

4.2.3 Pooling layers

The forward pass for a pooling layer is particularly simple in the no-laterals case (see Section 2.3). In this case, if $F^{(\ell+1)}_{f'r'c'}$ is the parent feature of a set of pools that we will call $\{H^{(\ell)}_j\}$, the bottom-up message to this parent, $F^{(\ell+1)}_{f'r'c'}$, is just

$$m_{\text{bu}}(F^{(\ell+1)}_{f'r'c'} = 1) = \sum_j \max_{s_j} m_{\text{bu}}(H^{(\ell)}_j = s_j), \tag{S25}$$

i.e., the bottom-up message for each feature is just the sum of the largest bottom-up messages of each of the pools that are part of that feature. The maximum is computed for each pool independently, so the message can be found efficiently.

We have observed that even if the model on which we are performing approximate inference has laterals, a no-laterals forward pass will produce a reasonably good solution in practice as long as the backward pass does use the laterals. Therefore, in most cases, we can just ignore the laterals if they exist and use the above update. If laterals are to be used during the forward pass, the update can be written as

$$m_{\text{bu}}(F^{(\ell+1)}_{f'r'c'} = 1) = \max_{\{s_j\} \in \text{const}(F^{(\ell+1)}_{f'r'c'})} \sum_j m_{\text{bu}}(H^{(\ell)}_j = s_j), \tag{S26}$$

where $\{s_j\}$ represents a pool configuration with s_j being the state of pool $H^{(\ell)}_j$ and $\text{const}(F^{(\ell+1)}_{f'r'c'})$ being the set of pool configurations that satisfy the lateral constraints imposed by $F^{(\ell+1)}_{f'r'c'}$ (see Section 2.3.2). Note that this is no longer a trivial maximization problem, since the states of all the pools are now laterally entangled and cannot be maximized independently. We dub this maximization subproblem "the laterals subproblem" and expand on it in Section 4.6. Because the lateral constraints might be loopy, the problem may be NP hard so that any procedure addressing it might result in a non-maximal pool configuration, and therefore in an approximate bottom-up message to the features on the next layer.

Because lateral connections tend to not have many loops, an intermediate solution between considering all the pools as independent and trying to address the full loopy problem is to introduce a few cuts in the constraint graph (for instance using the minimum spanning tree to to find which constraints

[12] If there are multiple pool states activating the same feature, they will belong to different pools.

can be removed with minimum impact) and then solve the laterals subproblem using max-product, which is exact on this simpler graph.

Layer $H^{(C)}$ is different from other pooling layers, but because it sits at the top of the hierarchy it does not need to pass on any bottom-up message so nothing needs to be done regarding this layer during the forward pass.

4.3 Hypothesis selection

After the forward pass, each bottom-up message to variable $H^{(C)}$ represents the detection level for each specific top-level feature (object) at each location. If the task at hand was simply to locate and classify the most salient object, finding the maximum over these messages would provide a solution to it. Instead, we will now proceed with a top-down backward pass of max-product belief propagation that provides assignments to every latent variable in the model, which at the bottom-most layers results in effectively reconstructing the most salient object, i.e., identifying the specific pixels that belong to it. Note that both the selection of the most salient object and its subsequent reconstruction are just part of the normal top-down max-product message passing in the graphical model and do not require to be introduced by hand in the process, see Section 4.4.2.

4.4 Backward pass

After completion of the forward pass, all the variables in the hierarchy have incoming bottom-up messages. However, we are looking for a concrete assignment of values to all the variables in the hierarchy that approximately maximizes the joint probability of the observation and latent states, i.e., an approximate MAP solution.

The backward pass proceeds top-down making such assignments, decoding the approximate MAP solution. We will apply the max-product rule to update the top-down messages, following a parallel update for all the variables of the same layer, updating each layer in turn from top to bottom. Immediately after the top-down messages for a layer have been computed, a hard-assignment for all the variables in that layer is made, which in turn influences the top-down messages being propagated further down. The hard-assignment sets each variable to the value with largest max-marginal (see Section 4.1). This results in a global approximate MAP solution.

Details of how this assignment is made for each layer follows.

4.4.1 Feature layers

Providing assignments to the variables in the feature layers during the backward pass is trivial. Since every feature layer is below a pooling layer which has already been hard-assigned (since the backward pass proceeds in a top-down fashion providing such hard assignments), a feature variable should be set to 1 if there are one or more pools above it activating and to 0 otherwise.

The final assignment at $F^{(1)}$ produces an edge map of the detected object where occluded or undetected edges are revealed. We refer to this object reconstruction at the edge level as *backtrace*.

4.4.2 Pooling layers

The first variable that requires a hard assignment is $H^{(C)}$. As mentioned earlier, the top pooling layer $H^{(C)}$ is different from other pooling layers in that it has a single variable which is never in the OFF state and does not have a top-down input. This variable is assigned to the state that maximizes the max-marginal, i.e., to activate the feature $F^{(C)}$ with the largest bottom-up message. In the case of a

tie, this is resolved uniformly at random[13]. This hard assignment results in a single top-level object to be reconstructed as a result of the top-down message passing, while the remaining objects are turned off.

Unlike the pool at the top layer $H^{(C)}$, which is never in the OFF state, pools at every other layer can be in the OFF state or not. This is determined by the state of their parent feature. If the parent feature of a pool is OFF (remember that each pool has a single parent), then the pool is in the OFF state.

If the pool is not in the OFF state and we are not using laterals during the backward pass, then each pool should be assigned to the state with the largest bottom-up message.

If the pool is not in the OFF state and we are performing a with-laterals backward pass, then we want all the pools that belong to the same parent to be in the joint set of states that maximizes the sum of bottom-up messages while being among those that satisfy the lateral constraints. I.e., each group of sibling pools $\{H_j^{(\ell)}\}$ is jointly set to the states

$$\{s_j\} \;=\; \underset{\{s_j\}\in \mathrm{const}(F^{(\ell+1)}_{f'r'c'})}{\operatorname{argmax}} \sum_j m_{\mathrm{bu}}(H_j^{(\ell)} = s_j), \tag{S27}$$

where $F^{(\ell+1)}_{f'r'c'}$ is the common parent of all the pools $\{H_j^{(\ell)}\}$. This is again the lateral subproblem of Section 4.6. If we already solved this problem during the forward pass, then there is no need to solve it again during the backward pass, since the bottom-up messages have not changed.

If the forward pass did not use the lateral constraints, we can still use them during the backward pass so that the final solution (assignment) does satisfy all the lateral constraints. It may seem at first that if we are going to have to solve the lateral subproblem during the backward pass anyway, we might as well always solve it during the forward pass. This would make the forward pass more accurate and the extra cost seems to be amortized during the backward pass. The catch here is that during the forward pass we do not know which features are going to be active in the final solution, so we would have to solve it for every possible feature. On the other hand, during the backward pass, it only needs to be solved a few times, for the features that we decide, layer by layer, that belong to the detected object.

4.4.3 External laterals

The pool choices $H^{(l)}$ that share a parent $F^{(l+1)}$ are entangled on the backward pass via the laterals. Pools that are cousins, sharing a grandparent $F^{(l+2)}$ but not a parent $F^{(l+1)}$, do not not constrain each other's choice during the backward pass. Consider modeling a line with a tree decomposition, i.e., dividing the line into line segments, which are further divided into subsegments, etc. One would expect the amount of distortion along the line to be uniform. However, at points along the line in which we jump from the group of pools sharing a parent to the group of pools sharing the next parent, discontinuities may appear, since there are no laterals among pools with different parents.

To fix this asymmetry additional laterals are added between cousin pools during the backward pass. This not possible during the forward pass as the grand-parents $F^{(l+2)}$ of a pool $H^{(l)}$ are unknown. If the computation was conditioned on which grand-parent feature was using that pool, the computation would get significantly slower as there can be many grand-parent features for a single pool.

The externals are included by adding additional factors ϕ_E to the factor graph that represent additional translational perturb laterals between cousin pools.

[13]We will always resolve ties at random instead of deterministically. This means that inference can produce different results when run multiple times, despite the deterministic message passing and scheduling.

4.4.4 The postprocessing layer

The postprocessing layer is the backward pass analog of the preprocessing layer. After all the discrete latent variables in the shape model (including $F^{(1)}$) have been assigned to some concrete value, the postprocessing layer (approximately) maximizes $\log p(X, Y|F^{(1)}) = \log p(X|Y) + \log p(Y|F^{(1)})$ wrt the canvas variables Y. In other words, each canvas variable (which corresponds to a pixel in the observed image) is assigned an IN or OUT type and a color (texture) type so as to be maximally compatible with both the image under analysis X and the edge map $F^{(1)}$. The IN or OUT type of these variables results in the *object mask* for the detected object. This mask covers the inferred surface of the detected object and includes occluded regions.

Since the object edges (which are already known at this point of the backward pass) are watertight, when $\gamma \to 0$, determining the IN or OUT type of all the variables in Y is trivial: variables in regions inside the object are set to IN and variables outside the object are set to OUT. This directly provides the mask of the object. To determine the exact state of each variable in Y that maximizes $\log p(X, Y|F^{(1)})$, one would need to resort to CRF energy minimization techniques (both X and $F^{(1)}$ are known, so the problem can be tackled by any pairwise MRF solver). Fortunately, in many applications, knowledge of the mask is enough.

Additionally, for a given (approximately) maximizing canvas state Y, obtaining $\log p(X, Y|F^{(1)})$ exactly is challenging because the cost of computing its log-partition function is exponential. This is not a problem for obtaining the maximizing Y since the log-partition function is constant wrt Y and can be ignored. But exactly computing $\log p(X, Y|F^{(1)})$, including the log-partition function, could be required if one had somehow obtained two different candidate shapes $F^{(1)}$ and $F'^{(1)}$ and wanted to check which one has the highest score $\log p(X, Y|F^{(1)})$.

A rough approximation to $\max_Y \log p(X, Y|F^{(1)})$ —to which we shall refer as the "postproc score"— can be obtained by summing the bottom-up messages incoming to active features in $F^{(1)}$. Each bottom-up message approximately quantifies the increase in $\max_Y \log p(X, Y|F^{(1)})$ that would be obtained if the corresponding variable in $F^{(1)}$ was turned on with respect to all the variables in $F^{(1)}$ being off. So summing the bottom-up messages corresponding to active features is a first order approximation to $\max_Y \log p(X, Y|F^{(1)})$ for an arbitrary $F^{(1)}$, up to a constant independent of $F^{(1)}$. Because the constant is independent of $F^{(1)}$, this approximation allows to compare the quality of two candidate shapes $F^{(1)}$ and $F'^{(1)}$ without computing a partition function. The disadvantage is that this rough approximation does not produce any IN-OUT propagation inside of the candidate shapes, so the approximate score is only related to shape, and not to appearance consistency within each of the regions defined by the shape.

It is worth mentioning that once all the latent variables have been selected to approximately maximize the final score of any object, this can be split into two components. The "postproc score" corresponds to the (approximate) computation of $\log p(X, Y|F^{(1)})$, whereas the "shape score" corresponds to the computation of $\log p(F^{(1)}, H^{(1)}, \ldots, F^{(C)}, H^{(C)})$. Observe that, ignoring border effects, the "shape score" given by Eq. S10 is a constant for each different top-level object, since the number and type of the active pools after the backward pass is constant and independent of the observed image.

4.5 Two-level model

The simplest usable version of the proposed hierarchy is a two-level mode with $C = 2$. In this case, inference is particularly simple and requires solving a single lateral subproblem per object category. Some of the experiments of Section 8 will use this hierarchy, simply referred to as "the two-level model", whereas others will use deeper hierarchies.

4.6 The lateral subproblem

The lateral subproblem is a standard MAP inference / energy minimization problem with discrete variables. On loopy graphs, the problem is known to be NP-hard [69]. Nonetheless, there had been numerous efforts trying to better approximate the solution more efficiently, such as [66, 67, 70, 71]. Most of the methods locally resemble the original loopy max-product BP algorithm but schedule the messages in different ways. In [68, 72], a unified framework had been established for understanding these method based on dual decomposition. And an enhanced version of this family of algorithms was proposed in [68], called Subproblem-tree Calibration (STC), that we use in practice.

The basic idea of dual decomposition is to construct an upper-bound of the MAP inference objective function by maximizing it for each lateral independently. We then minimize this upper-bound by re-parametrizing the laterals through exchanging messages among them to enforce consistency. Different algorithms differ in the way they schedule these messages. STC chooses to iteratively take different spanning trees of laterals and ensure local dual optimality by calibrating these large trees, such that a globally consistent and high quality solution would emerge. The reader is referred to [68] for more details of the algorithm.

4.7 Images with multiple objects: parsing

The model as presented so far generates one object at a time. In general, multiple objects can be present in a single image. One could think of two alternative approaches to use it to understand complex, multi-object images:

(a) Serial, closed-loop: Use a forward and backward pass to detect the most salient object in an image, their borders, and mask, remove it from the image, and repeat. Each repetition requires repeating the forward pass and the backward pass.

(b) Parallel, open-loop: Use a single forward pass to detect all the objects that might exist in the image at once using both a threshold and non-max suppression on the bottom-up messages to $H^{(C)}$. Thresholding removes unlikely candidates, whereas non-max suppression is used to avoid introducing the same candidate multiple times with small translation (or otherwise) differences. Then run a backward pass for each of the selected candidates. Provided a scoring function that is able to compute how likely it is for a candidate subset to be the explanation of an image, find the candidate subset that maximizes it and return it as a parse of the image.

In this work we will follow the second approach. Two advantages of the second approach are that it requires a single forward pass and that it is less greedy so that we can hope for a better solution to be recovered.

The non-max suppression step is a shortcut for the explaining away that would naturally appear in the model if we were to perform full-fledged inference: After the forward pass, multiple alternative top-level variables, each corresponding to a given category and position will show a high score. This is because top-level explanations that only differ in position by a small margin can produce the same instance in the image under different pool configurations at the layers in-between. If we clamp one top-level variable (corresponding to a given category and position) to 1 (meaning that it is present), additional max-product message propagation will turn off the other alternative nearby top-level variables, because the instance in the image has already been explained and there is no incentive for max-product to activate additional causes for the same observation. Note that this explaining away and suppression of alternative explanations occurs as a result of the model definition and structure and requires no additional parameters.

The explaining away effect can be mimicked without additional propagation by local non-max suppression. This requires a radius parameter. This should be set based on the architecture and not on training data. This parameter captures the intrinsic resolution provided by a given RCN parameterization and not the spacing between instances in training data. Thus, when letters in a training set for

CAPTCHA are very spread, we do not increase this value (even though we could without decreasing performance), and thus can properly generalize to test cases in which the spacing between letters is reduced. To see this in practice, check Section 8.4.3.

4.7.1 Scene scoring function

First, we describe the scoring function over parses of a given scene. The next section will then describe how to optimize this score efficiently.

As explained in Section 4.4.4, it is possible to use the sum of the bottom up messages to $F^{(1)}$ to approximately compute the "postproc score" of the shape described by $F^{(1)}$. The "postproc score" lacks the Occam razor effect that the "shape score" given by Eq. S10 provides to make more complex objects less likely. Our scene scoring function uses one term that approximates the "postproc score" of a complex shape composed of multiple objects and adds a specific term penalizing for complexity.

Following the approach (b) outlined above, the result of the single forward pass and multiple backward passes are a single set of bottom-up messages B (where B is three dimensional array with a one-to-one correspondence with the elements of $F^{(1)}$) and several sets of top-down binary activations $\{T^{(m)} : m \in 1, 2, \dots M\}$, each corresponding to a different candidate m (where each $T^{(m)}$ is again a three dimensional array with a one-to-one correspondence with the elements of $F^{(1)}$). A parse is a binary vector v of length M and elements $\{v_m \in \{0, 1\}\}$ that specifies which candidates are included in that parse.

If we combine the top-down activations $T^{(m)}$ of the candidates that are active in parse v by ORing them, we can write the combined top-down activation as

$$T = \min\left(\sum_m v_m T^{(m)}, 1\right) = \sum_m v_m T^{(m)} - \max\left(0, \sum_m v_m T^{(m)} - 1\right), \tag{S28}$$

where the first equality is an intuitive way to define T and the second one explicitly decouples the total, aggregated activations from an overlap penalty term. The min and max functions between a matrix and a scalar have the usual definition, they are applied elementwise, comparing each element of the matrix with the scalar and returning the result as a matrix with the same arrangement.

Using the joint activation T, we can compute the approximate joint "postproc score" of a parse as

$$\sum_{frc} B_{frc} T_{frc} = \sum_{frc} B_{frc} \sum_m v_m T^{(m)}_{frc} - \sum_{frc} B_{frc} \max\left(0, \sum_m v_m T^{(m)}_{frc} - 1\right). \tag{S29}$$

However, empirically we find that when scoring parses, this "postproc score" can be improved with two small modifications to the second term (the one taking care of the overlap penalty). First, instead of penalizing with B_{frc}, we use a constant term $\text{overlap}_{\text{penalty}}$. Second, instead of computing the overlap between different $T^{(m)}$ directly, we compute it between dilated[14], flattened[15] versions $\text{dil}(T^{(m)}_{\text{flat}})$ of the activations, so that if the thin contours represented by $T^{(m)}$ are close to overlapping even in different

[14]The result of dilating a binary matrix A with radius d is another binary matrix B in which entry B_{rc} equals 1 iff $A_{pq} = 1$ for some pair p, q fulfilling $|p - r| < d$ and $|q - c| < d$. In words, every active element of A becomes surrounded by active elements in B. In the case at hand, the dilation is applied to each channel f.

[15]The flattened version $T^{(m)}_{\text{flat}}$ of $T^{(m)}$ is a two-dimensional binary array which corresponds to collapsing all the channels of $T^{(m)}$ by ORing them.

channels, they will overlap in the dilated version. The corrected joint "postproc score" of a parse is

$$\sum_{frc} B_{frc} \sum_m v_m T^{(m)}_{frc} - \sum_{rc} \text{overlap}_{\text{penalty}} \max\left(0, \sum_m v_m \text{dil}(T^{(m)}_{\text{flat } rc}) - 1\right). \tag{S30}$$

We can finally write the *scene score* of v by combining the above "postproc score" with a complexity penalty:

$$\text{ss}(v) = \sum_{frc} B_{frc} \sum_m v_m T^{(m)}_{frc} - \sum_{rc} \text{overlap}_{\text{penalty}} \max\left(0, \sum_m v_m \text{dil}(T^{(m)}_{\text{flat } rc}) - 1\right) - \delta \sum_m v_m \sqrt{N^{(m)}_{\text{pools}}}, \tag{S31}$$

where $N^{(m)}_{\text{pools}}$ is the number of active pools at the lowest level for candidate m (which is a constant for each top-level feature or object category in the two-level case) and δ is a parameter controlling the trade-off. Observe that the last term of this scoring function has the same role as the "shape score" for individual candidates: it penalizes more complex candidates consisting of a larger number of pools. The square root could be replaced by any exponent between 0 and 1, which can be determined empirically on a validation set. We refer to this parameter as the "pool reweighting exponent"

4.7.2 Optimization of the scene scoring function

We are interested in computing the best parse for a given scene according to the scene score (Eq. S31). I.e., we want to compute $v^* = \text{argmax}_v \, \text{ss}(v)$. This problem is exponentially complex in the length of vector v, so we tackle it using an approximate dynamic programming technique that only has quadratic complexity and then show how a further mild approximation yields a linear time algorithm. The approximation used in the dynamic programming turns out to be exact in several real world scenarios, so we have often found the results of the next algorithm to be optimal.

In order to develop a dynamic programming algorithm for this problem, consider first that a topological ordering exists among the candidates that can be included in a parse. For instance, when the candidates in a scene are symbols arranged in the form of a text (such as characters forming a CAPTCHA), a natural topological ordering could be the left to right. In complex scenes with multiple candidates (for instance, several objects randomly scattered on the floor), a left-to-right ordering does not make much sense. In that case, we can select a fixation point and order the candidates according to the distance from the fixation point to the most distant point of each of them. The ordering should aim at keeping far apart objects that are topologically far apart. Thus, each of the candidates is assigned a different ordering number $m \in 1 \ldots M$.

The key function that we want to compute (or approximate) is the prefix parse prefix$[m]$, which is defined as follows:

$$\text{prefix}[m] = [\underset{x_{m-1}}{\text{argmax}} \, \text{ss}([x_{m-1}, 1, 0_{M-m}]^\top), \; 1], \tag{S32}$$

where x_{m-1} is used to denote a vector containing $m-1$ binary variables and 0_{M-m} is a vector of zeros of length $M-m$. Observe that the prefix parse prefix$[m]$ tells us what is the best parse that contains candidate m and does not contain any of the candidates with ordering number above m. Also, note that prefix$[m]$ is a vector of length m, with a 1 always being its last element. For convenience, we additionally define prefix$[0]$ to be the empty vector.

Each prefix parse prefix$[m]$ has an associated prefix score, which is computed by swapping the argmax with a max.

$$\text{ps}[m] = \max_{x_{m-1}} \text{ss}([x_{m-1}, 1, 0_{M-m}]^\top). \tag{S33}$$

It should be obvious that the optimal parse v^* is the one given by prefix$[m]$ when evaluated at the m that maximizes ps$[m]$. In words, the best parse overall should be the (possibly empty) best prefix parse, since each of them is only restricted to have as last active candidate each of the available candidates (except for the empty prefix parse). And obviously the best parse must either be the empty parse, or have one of the existing the candidates as the last one.

Being able to compute all prefix parses exactly would solve the problem optimally. We turn now to an efficient (but approximate) recursive way to compute each prefix parse based on the previous prefix parses. This method is exact under some conditions often fulfilled in practice that will be discussed below.

We want to compute the prefix parse prefix$[m]$ for some value m when the previous m prefix parses (prefix$[i-1]$ for $i \in 1 \dots m$) are known. To do that, we evaluate a set of prefix parse proposals $\{\text{propos}[i, m]\}_{i=1}^{m}$ and take the best. The proposals are

$$\text{propos}[i, m] = [\text{prefix}[i-1], 0_{m-i}, 1]^\top, \tag{S34}$$

i.e., we combine each of the m previously existing prefix parses with the current candidate to form the proposals. Then we select as prefix parse the best scoring proposal

$$\text{prefix}[m] \simeq \text{propos}[\underset{i \in 1 \dots m}{\text{argmax}}\, \text{ss}([\text{propos}[i, m], 0_{M-m}]^\top),\ m]. \tag{S35}$$

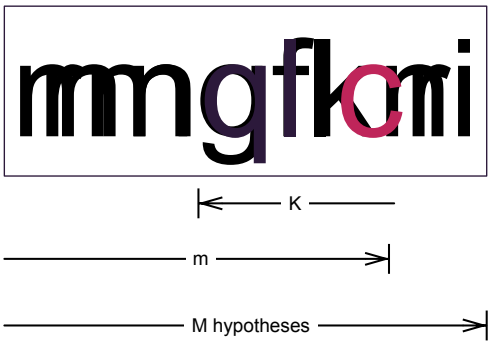


Figure S5: Illustration of the parsing algorithm. The dynamic programming algorithm exploits a topological ordering of the hypotheses masks to find a solution in linear time.

This dynamic programming algorithm is approximate because Eq. S35 is not exact: if the candidate under consideration m overlaps with candidate $m-2$ or earlier, the proposals might or might not[16] contain the optimal prefix parse as defined in Eq. S32. If they don't, then the prefix parse and score stored at prefix$[m]$ will be approximate.

The optimization algorithm computes Eq. S35 for every m sequentially, in a recursive fashion. The scene score function is therefore evaluated $\mathcal{O}(M^2)$ times, which makes the algorithm quadratic. It might seem that evaluating the scene score is itself $\mathcal{O}(M)$, but since each evaluation corresponds to adding the current candidate to a previously scored prefix, caching previous results makes each evaluation of ss$(\cdot)$ be actually $\mathcal{O}(1)$ when used as part of this algorithm.

[16]The condition of non-overlap between parse m and $m-2$ is sufficient but not necessary for the exactness of the algorithm. Additionally, we have observed in practice correct parsing under severe overlap conditions.

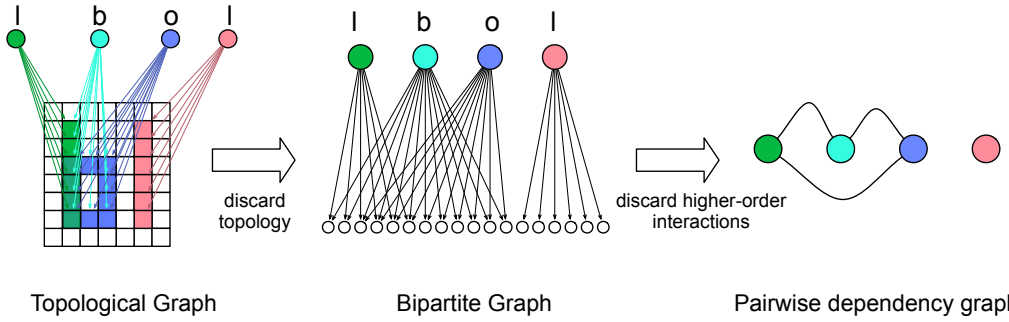


Figure S6: Different formulations of the parsing problem using different levels of information. Our DP algorithm uses the formulation on the left that exploits the generated object masks and their topology to resolve high-order interactions between object hypotheses. Discriminative models are restricted to operating using dependency graphs in the label space.

We can further reduce the complexity of the algorithm by making a practical observation: When we are computing prefix m, proposal propos$[i, m]$ corresponds to assuming that no candidate between i and m belongs to the best parse. This is more and more unlikely to happen as i and m space further apart. Therefore, it makes sense to only consider a finite lookback window, (see Fig. S5), and for the estimation of each prefix m compute only the K candidates for which $m - K < i \leq m$. This makes the whole parsing algorithm $\mathcal{O}(M)$.

It is worth noting that this dynamic programming algorithm makes use of the generative model by utilizing the topology of the generated masks to resolve high-order interactions between object hypotheses. Fig. S6 illustrates how different hypotheses generated after the forward-backward passes interact; the image of the "b" in Fig. S6 can be explained using a combination of "l" and "o". This represents a high-order interaction between these hypotheses, and the information required to resolve these interactions is available in the topological graph formed by the hypotheses and their masks. This high-order information is lost if the topological graph is converted to a dependency graph in the label space alone. This demonstrates another advantage generative models have over discriminative models when it comes to scene parsing. Discriminative models are restricted to using a dependency graph in the label space, whereas generative models can make use of the rich information of the generated object masks to resolve high-order interactions.

5 Learning

We next address learning a hierarchical model from data. The algorithm proceeds layerwise, in bottom-up order. For example, when the depth C is 4, we learn the parameters for feature layer $F^{(1)}$, followed by $F^{(2)}$, $F^{(3)}$, and $F^{(4)}$. The bottom-up messages into $F^{(1)}$ are computed by the preprocessing layer, so are not learned.

There are two components of the model to be learned: features and lateral connections. Features are learned from a training set with images preprocessed to extract the contours as described in Section 4.2.1, using unsupervised dictionary learning and sparse coding (Section 5.1). Lateral connections for pooling layers are learned from the contours of the input, and their perturbations are parametrically generated using the perturbation model (Section 5.2). Since the layers are learned independently, individual pieces can be replaced.

The learning relies primarily on unlabeled data. Labels are needed only for learning top-level features, and this step needs one labeled image per classifier feature (class).

The algorithms below also make one further assumption about the hierarchy, which is that there is a bijective mapping from each feature at a given level, to a pool at the level above that is said to be

"centered" at that feature. This is true by definition in the translational case, and the general pool learning algorithm also enforces this assumption.

5.1 Feature learning

At high level, feature learning proceeds as outlined in Fig. S7 (best viewed in color). This figure illustrates the scenario in which $F^{(1)}$ has already been learned and we are learning $F^{(2)}$. The same procedure can be applied recursively to learn any number of layers. When a training image (in this case, a star) is presented to the network, we try to express it as a combination of existing, already learned $F^{(2)}$ features (red, green, and blue in the figure). However, parts of the image (in this case, the top right corner of the star) will not be captured by the existing $F^{(2)}$ features. Those will have to be expressed using $F^{(1)}$ features (pink and yellow in the figure). These results in a parse (a.k.a. sparsification) of the current images composed of $F^{(2)}$ features and $F^{(1)}$ features. The active $F^{(1)}$ features that are not part of any active $F^{(2)}$ feature can then be grouped based on contour continuity to form new $F^{(2)}$ features (i.e., the pink and yellow features will be grouped to create a new $F^{(2)}$ features, dashed in the figure). Repeating this process for all training images produces a set of $F^{(2)}$ features, which can then be pruned based on a trade-off between number of active features and reconstruction error. We now delve into the details of this procedure.

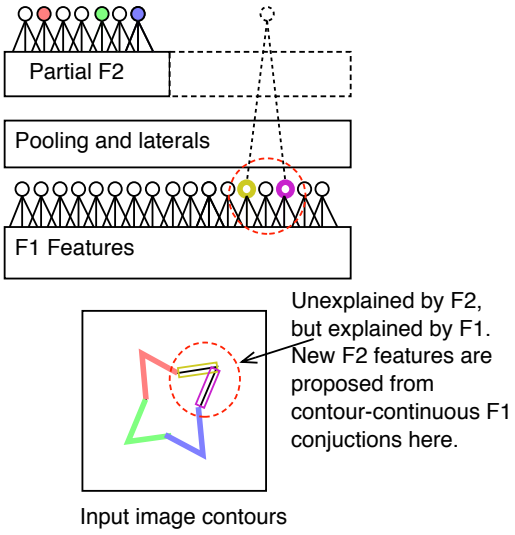


Figure S7: Learning features at the second feature level. Colored circles represent feature activations. The dotted circle is a proposed feature.

5.1.1 Sparsification

As described above, an important piece of the feature learning algorithm is *sparsification* (sparse coding). The goal of sparsification is to compute a compact representation S of an input image based on a set of reusable, translatable features. A sparsification can be represented as $S = \{(f_1, r_1, c_1), \ldots, (f_K, r_K, c_K)\}$ where each f_k is a feature at level-ℓ.

Our sparsification algorithm shown in Algorithm 1 aims to minimize the following cost function over sparsifications at level ℓ:

$$J_{\text{sparse}}(S; X) = \alpha|S| + \log P(X|S), \tag{S36}$$

where the $P(X|S)$ is just the conditional distribution of our generative model over the edge map given that exactly the features in S are on at level ℓ. The approach is similar to matching pursuit [73],

but uses the RCN to allow distortion of features in the matching step. Other hierarchical learning approaches such as [74] do not allow distortion (pooling) while composing features, making this a distinctive feature of RCN. At each iteration, the algorithm maintains the current sparse representation and the remaining unexplained portion of the lowest-level activations (the bottom-up message to $F^{(1)}$). It then does a truncated forward pass up to level $C-1$ and picks the most active level-C feature. It then does a backward pass to determine the corresponding $F^{(1)}$ features, and suppresses all activations within a neighborhood of each such $F^{(1)}$ feature. The process is repeated until the entire image is explained or no good matches remain.

Algorithm 1 Greedy sparsification

procedure SPARSIFY($m_{\text{bu}}(F^{(1)})$, `RCNModel`)
  `Sparsification` $\leftarrow \emptyset$
  `Unexplained` $\leftarrow m_{\text{bu}}(F^{(1)})$
  while True **do**
    `Messages` = TRUNCATEDFORWARDPASS(`Unexplained`)
    `MostActiveFeature` = $\operatorname{argmax} m_{\text{bu}}(F^{(C-1)})$
    if $m_{\text{bu}}(F^{(C-1)})$(`MostActiveFeature`) $<$ `Threshold` **then**
      return `Sparsification`
    end if
    `Sparsification` $\leftarrow$ `Sparsification` $\cup$ {`MostActiveFeature`}
    `ExplainedLocs` $\leftarrow$ BACKWARDPASS(`Messages`, `MostActiveFeature`)
    `Unexplained` $\leftarrow$ SUPPRESSAROUND(`Unexplained`, `ExplainedLocs`)
  end while
end procedure

5.1.2 Intermediate layer feature learning

Intermediate feature layers in the RCN model are learned from unlabeled training images. The feature layer at level ℓ consists of N features (where N must also be determined), and each feature consists of a set of child pools, which we will write as $\{(f_1, r_1, c_1), \ldots, (f_M, r_M, c_M)\}$, where (f_m, r_m, c_m) denotes the pool at the child level $H^{(\ell-1)}$ that is centered around that $F^{(\ell-1)}$ feature.

A cost function $J_{\text{sparse}}(S; D, X)$ over sparsifications of an image at level ℓ was defined in Section 5.1.1. Note that we have made the dependence on the feature dictionary D explicit. Now, given a dataset $\mathcal{X}$ of images, define the cost function for a dictionary to consist of a term for how well we can sparsify each image, and a term penalizing dictionary size:

$$J(D; \mathcal{X}) = \beta|D| + \frac{1}{|\mathcal{X}|} \sum_{X \in \mathcal{X}} \min_{S} J_{\text{sparse}}(S; D, X). \tag{S37}$$

We also assume that the feature size for intermediate layers is restricted to some range (e.g., each level 3 feature consists of between 3 and 7 pools). Let $\mathcal{D}$ be the set of dictionaries of such features. The overall learning problem for a single feature layer is then $\min_{D \in \mathcal{D}} J(D; \mathcal{X})$.

Our learning procedure greedily searches for an approximate solution to the above problem. The outer loop is described in Algorithm 2. It constructs a dictionary by adding one or more features on each iteration. It also maintains partial sparsifications of each training image in terms of the features learned so far. The individual steps work as follows:

- PROPOSEFEATURES picks a random image, and then looks at the child-level representation of the unexplained portion of that image. It then proposes a set of children to group into a feature. Rather than propose arbitrary sets of children, which are unlikely to lead to useful features, we form an adjacency graph over the child features and propose a connected set in that graph.

Algorithm 2 Feature learning outer loop

```
1: procedure LearnDictionary(𝒳, HierarchyBelow)
2:     Dict ← ∅
3:     Sparsifications ← (∅, ..., ∅)
4:     while True do
5:         Candidates ← ProposeFeatures(Dict, Sparsifications, 𝒳)
6:         Chosen ← PickGoodCandidates(Candidates)
7:         Dict ← Dict ∪ Chosen
8:         Sparsifications ← UpdateSparsifications(Sparsifications, Chosen, 𝒳)
9:         if Chosen = ∅ then
10:            return Dict
11:        end if
12:    end while
13: end procedure
```

Furthermore, we use a stratified approach, where features of the largest size are proposed first until no longer useful, then features of the next largest size, and so on.

- PickGoodCandidates computes how often each candidate would be used, then picks the ones that would be used in some minimum fraction of the dataset, and removes duplicates (pairs of features that have significantly overlapping sets of realizations given the distortion model at lower levels).
- UpdateSparsifications recomputes the sparse representations for any images that use one of the newly added features.

We follow the above procedure using 10k images taken from the SHREC 3D dataset [75]. Fig. S8(b) shows the features learned at level 2 by the above algorithm. Each F2 feature was constrained to consist of exactly three F1 features. As can be seen, the algorithm learns oriented edges and some curves.

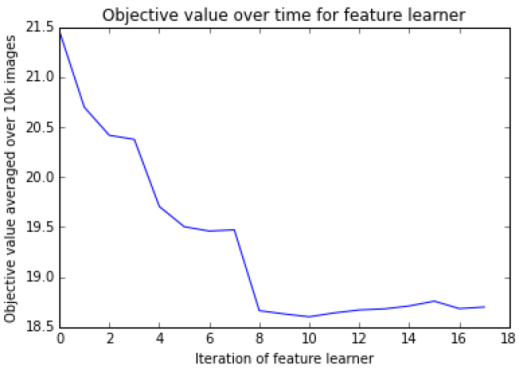
Figs. S8(c)-(e) show the features learned at level 3. Each F3 feature was constrained to consist of between three and five F2 features. As can be seen, at each size we initially learn oriented edges, and shift over time to curves and corners. Fig. S9 shows the sparsifications of a few test images at various intermediate points in the algorithm. The sparsifications gradually become more complete as more features are added. Finally, Fig. S8(a) shows the progress of the objective function after each iteration of the learner outer loop, demonstrating that the greedy algorithm does decrease this quantity.

5.1.3 Top-level feature learning

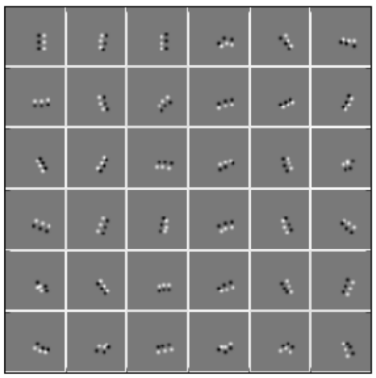
The classifier features at the top level of the hierarchy are learned from labeled training images. This step is nonparametric, creating a new classifier feature per training image. While this may seem expensive, our experiments demonstrate that the invariances in our model allow large classes of shape and appearance variations to be captured in many cases without a prohibitively large number of classifier features. Thus, given a large training set, we can generally train on a subset of the images, chosen either randomly or in some more informed way, as we do in our MNIST experiments in Section 8.7. Limiting or pruning the classifier feature set is nevertheless an important direction for future work.

The top-level feature learning is then a special case of Algorithm 2 in which:

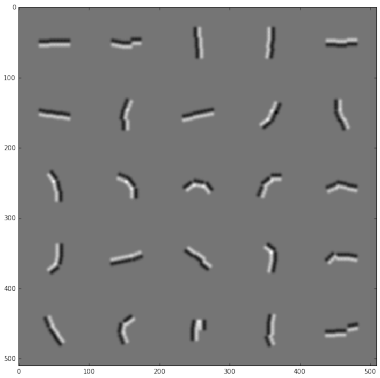
- There is only one outer loop iteration.
- The proposal step returns one classifier feature per labeled image, computed using Algorithm 1.
- The candidate filtering step accepts all candidates.



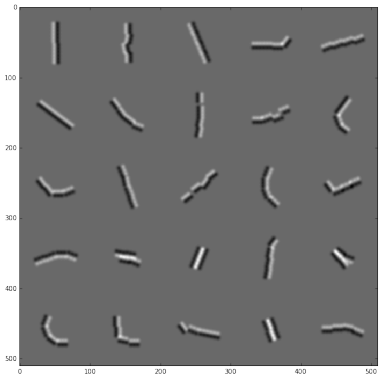
(a) Evolution of objective (Eq. S37) over time



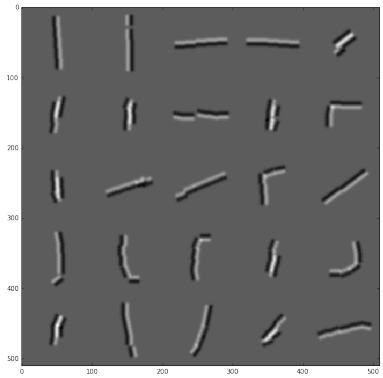
(b) F2 features



(c) F3 features (3 F2 features each)



(d) F3 features (4 F2 features each)



(e) F3 features (5 F2 features each)

Figure S8: (a) Evolution of the objective function (Eq. S37) on the training set of 10k images as the feature learner progresses. After each iteration, pass-through copies of the lower level features were concatenated to the features learned so far, and the images were then sparsified. As a result, the images are well represented at each iteration, and the cost function ends up being approximately equal to the average representation size per image plus a term penalizing the dictionary size. (b-e): Features learned at levels 2 and first 25 features of each size learned at level 3 by the greedy learning algorithm. Visualized is the projection of each feature down to the pixel level, where white and black pixels represent the bright and dark side of the feature respectively.

5.2 Learning laterals

The other component of the model is the lateral connections between pools. Learning the lateral connections involves (1) learning the structure of the lateral graph (which pairs of pools have a lateral constraint) and (2) learning the lateral factors that specify the pairwise compatibility between the pool members.

The lateral graph structure (the connectivity between pool pairs) is learned from the contour connectivity of the object in the input image — pools with features that are adjacent in the input contours are connected with each other. This process is applied recursively in the hierarchy to learn the lateral connections between pool pairs at each level. Additional edges are added to enforce that pools that are close in pixel space are also close in the lateral graph (this includes, as a special case, adding edges to make the lateral graph connected, as required for categories with multiple spatially separated parts such as lowercase "i"). We refer to these as "under-constraint edges".

The lateral factors specify the compatibility between members of pools that are adjacent in the lateral graph. This is parametrically populated according to the translational perturbation model described in Section 2.3.2. The flexibility of these factors is controlled using a single parameter per level, and those parameters are set using cross validation. The translational constraints give reasonable results

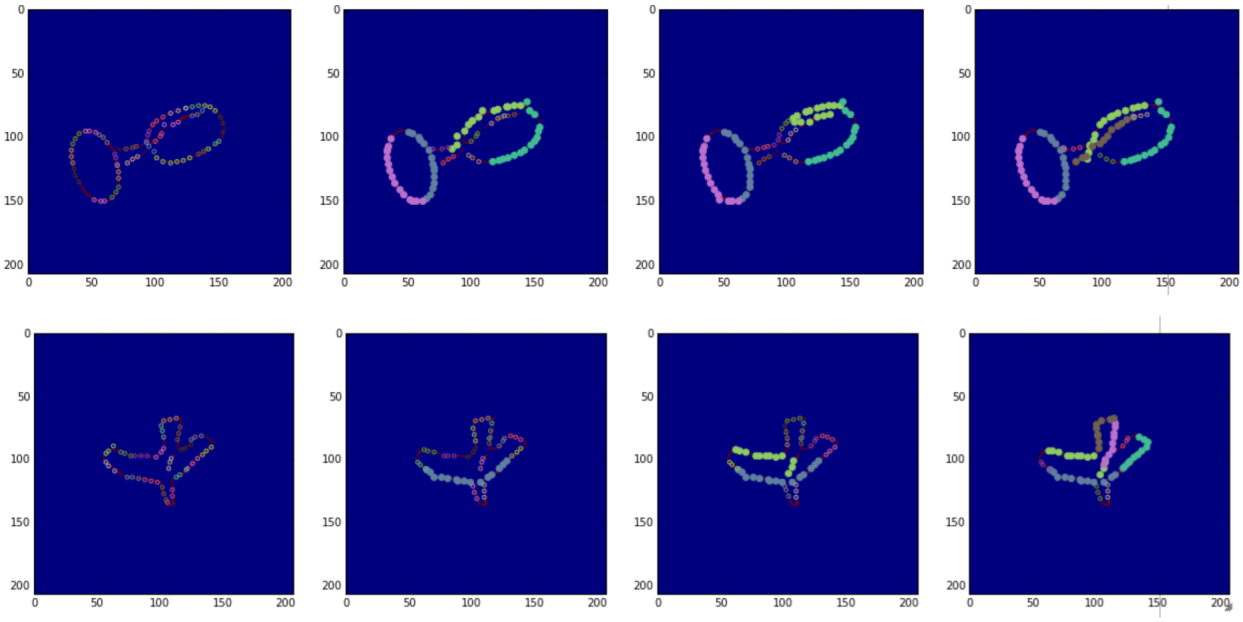


Figure S9: Snapshots of the sparsifications of training images at various intermediate points during Algorithm 2. In each figure, solid circles denote F1 features belonging to F3 features and each color denotes one instance of a particular feature. Unfilled circles denote F1 features not yet explained.

in our experiments, but additional selectivity could be obtained by learning non-uniform factors. This is an important area of future work.

6 Related work

Compositional models have a rich history [24, 30, 39, 42, 76–79]. In [24], Zhu and Mumford set out an ambitious agenda for compositional models constructed analogously to the grammar models in language. Concrete instantiations of these models [34,39] have had notable success in computer vision. Despite the similarities in their pictorial representations, grammar-based models are semantically different from probabilistic graphical models, which is the representation we use. Grammars exclude the sharing of sub-parts among multiple objects or object parts in a parse of the scene [32], and they limit interpretations to single trees even if those are dynamic [80]. Our graphical model formulation makes the sharing of lower-level features explicit by using local conditional probability distributions for multi-parent interactions, and allows for MAP configurations (i.e, the parse graphs) that are not trees. This graphical model representation allows greater modularity and the use of generic inference mechanisms that rely on local message passing. Although [24] anticipated the need for lateral constraints, practical implementations have focused on AND-OR graphs that are tree-structured due to the difficulty of inference in loopy structures. A closely related family of models called *recursive compositional models* [25,79,81] use lateral constraints, but they lack a pooling structure that gradually builds invariance level by level, choosing instead to model transformations globally and parametrically at every level. When the transformations from different tree branches overlap, their interactions are not explicitly modeled, leading to unclear generative semantics; shape and appearance are not factorized. The lateral connections require a particular triplet arrangement, which leads to long distance effects not being propagated by their custom inference mechanism because of the greedy decisions made at each triplet.

Composition Machines (CM) [32] have similar high-level inspiration as RCNs and they recognized many of the shortcoming of grammar-based models. From a model perspective, the functionality of the feature and pooling layers of RCNs is collapsed in the case of CMs into a single layer of variables called “bricks”. This drastically increases the state space of CMs’ variables, making its

representation less efficient; a single brick is a multinomial variable that needs to capture all the possible expressions of a feature, including all the pooling states of its elements and its coordination. In RCNs, a feature only specifies which pools constitute it, while their variability is delegated to separate variables – the pool themselves. The proposed coordination of the pools in RCNs through pairwise lateral compatibility functions is also more efficient than the "composed distributions" of CMs that evaluate the compatibility of a joint assignment of pools. In terms of inference, RCN uses the well-known max-product message passing method to find a MAP "parse" of the observation in terms of the model (requiring a bottom-up pass and a top-down pass), whereas CMs use a greedy bottom-up heuristic that selects the parse at each level without integrating any top-down information. Finally, no learning algorithm has been proposed to discover the connectivity of CMs, and this is necessary to scale it to problems beyond its original domain of application.

Very few works have pursued shape and appearance factorization for image recognition [63, 64]. Our shape and appearance factorization is different from that of [37], which splits an image into regions of contours and textures, but without the context of a higher-level object-based shape model as in RCN. The advantage of combining shape and appearance cues for semantic part segmentation was recently demonstrated [56]. Although this model has three different types of edge polarities, the model does not factorize shape and appearance or propagate local border ownership likelihoods to come up with a global object-based assignment. Instead, the edge polarities are assigned greedily using local cues.

The importance of unifying recognition and segmentation has been widely recognized [30, 82, 83]. Due to the difficulty of inference in generative models, [30] focuses on combining discriminative models trained for bottom-up inference with top-down generative models. Although several proof-of-concept examples of text detection and recognition were shown, these relied on the characters being separated well using Niblack thresholding. Also, the inference pipeline relied on a combination of AdaBoost classifiers and the solution of stochastic partial differential equations. No recognition accuracy was reported for large-scale tests, and no follow-up work on these ideas has been published, perhaps due to the complexity of the inference mechanisms. Tree structured graphs were used in [82], and the graph transformer method of [83] relies on a feed-forward discriminative approach that is trained extensively on the character strings (which need to be parsed).

Despite the advantages apparent in the compositional generative approach, the current instantiation of RCN has some limitations compared to deep neural networks (DNNs) for vision tasks. First, because RCN stores prototypes of every training sample in its penultimate layer (for classification), scaling to very large datasets, like ImageNet and other object recognition benchmarks where CNNs (e.g. AlexNet [65]) are state-of-the-art, may be intractable. A possible method to address this is to store flexible, probabilistic representations of training samples in the classification layer; i.e., merge multiple prototypes in a Nave Bayes fashion. We have explored such techniques, but they were not required for the problems presented in this paper. Moreover, our experiments with the large ICDAR dataset shows RCN as-is can achieve results comparable to DNNs by using a subset of the available training data.

Discriminative models have an additional advantage over generative counterparts on standard vision benchmarks because they are able to utilize scene context and backgrounds. By optimizing directly for the task at hand, such as object recognition, DNNs are able to exploit additional contextual information in the images. To do this, generative models like ours call for modeling not only the objects but also the background scenes in a generative manner. This is a current direction for improving RCN.

In the current instantiation of RCN, some of the parameters for preprocessing images are adjusted manually. DNNs, on the other hand, learn these automatically. These parameters could be learned from data and, more importantly, adjusted dynamically during inference, which is something we are addressing as part of future work. Additionally, current DNNs are able to train in the presence of noise, while the current instantiation of RCN requires training on clean data samples. Although the ability to generalize from training on clean data to testing in noisy domains is desirable, this can be

undesirable in applications where clean data samples are not readily available. Deep-learning models, on the other hand, do not have this drawback; they train explicitly on the domain where they will be tested, noisy or clean. We are currently exploring possible ways to address this limitation by implementing training schemes with gradient-based learning and message-passing mechanisms.

Recently, the Attend, Infer, Repeat (AIR) model [58] incorporated a simple form of compositionality into a neural network using a spatial attention window. Despite a high-level similarity, it is noteworthy that the top-down object-based attention in RCN is very different from the spatial attention window in AIR. RCN is compositional at the object level, and it factorizes the object from the background, which enables it to disentangle objects that are highly overlapping using top-down object-based attention. In AIR, each glimpse is modeled as a standard VAE which, as we demonstrate in our experiments in section S10, does not perform well when there is noise or clutter within the attention window. Theoretically, it can be expected that the current instantiation of AIR requires good separation between the objects in an uncluttered setting for its operation. Moreover, AIR has to be trained with multiple objects in the scene, and it is unclear if it can generalize to a setting where significantly more objects are present in the test image compared to the training images.

7 Neuroscience guidance

Several representation choices in our model were guided by neuroscience, without which it would have been very difficult to arrive at the combination of representational choices manifest in the model. We specifically look to representations and information processing in the ventral stream, the series of cortical regions that subserve object recognition in mammals [84, 85]. This implies a hierarchy of processing stages that encode image content increasingly in successive levels. The model of alternating hierarchical layers of feature detection and pooling is inspired by the early studies of cat visual cortex by Hubel and Wiesel [86]. RCN builds on their model of simple and complex cells, as do the HMAX and CNN models. An important insight for our model was that complex cells in vision regions of cortex are not simply local feature detectors, but rather serve the purpose of interpreting surfaces and extracting surface boundaries.

Although the pooling structure is represented as hard-coded translation pooling in the current RCN model, an important self-imposed restriction is for the pooling be local and non-parametric, based on the idea that ultimately the pooling structure itself should be learnable from temporal proximity [87], with very little built-in knowledge about the parametric form of the transformations. In contrast, global parametric transformations used in many compositional models try to achieve invariance to transformations in one step, but this makes message-passing difficult because of the lack of structure within the model. This is reflected in our empirical results in Table S5 where reducing the pooling and lateral flexibility resulted in better accuracies when more training examples are available.

Lateral connections in the visual cortex are well studied [17, 88, 89] and are hypothesized to enforce contour continuity. Local horizontal connections in vision regions support contour completion relative to local receptive fields [11], whereas long-range and higher-order interactions in the lateral connections may go beyond a local pairwise association field between features [89]. In our model, coordinating receptive fields at different hierarchical levels is achieved by keeping separate copies of lateral connections in the context of different higher-level features; superposing these different lateral connections by marginalizing over the parent features would give rise to a pair-wise association field. This idea is similar to the state copying ideas proposed in biologically inspired sequence learning models [90] and in recent work on higher-order temporal networks [40]. The interplace of lateral connections and pooling was inspired by the observations about invariance and selectivity dilemma as described in [35]. This architecture choice also allows us to utilize laterals in future work towards incorporating temporal patterns and dependencies into RCN as part of a general AI system.

Psychophysical and anatomical evidence exist for segregated yet interacting processing streams for contours (form) and surfaces (color) [20,22,91–93], motivating the shape and appearance factorization

in RCN. We iterated through several graphical model structures, with different tradeoffs for the interaction between contours and surfaces, before arriving at the structure proposed in this paper. One of the options we considered was to treat surface features just like contour features. However, this would require pooling the local translations of surface features, resulting in unnecessary computations and indeterminism during inference because surface patches with identical appearance can effectively swap their locations. An important consideration for the contour-surface interaction is that it is not possible to determine from local cues whether the contour belongs to the surface or not – i.e., the problem of border ownership [94]. Without having a hypothesis about the spatial layout of the object that is being considered, it is non-trivial to perform local inference on contour-surface interactions; considering all the possible object layouts for a given local contour-segment is an intractable inference problem. Neuroscience studies have discovered border ownership selective neurons in early visual cortex, preferring a given figure to be on one side of a border or the other [23, 95]. Borrowing an elegant solution found in neuroscience to have two copies of every contour neuron [19, 94], one representing each side of the border ownership, enables the RCN inference algorithm to propagate local uncertainty to arrive at a globally coherent solution. [95] suggest feedback from higher regions of visual cortex (IT) make earlier cells (in V2) border ownership selective. This is consistent with the idea of enforcing contour continuity and settling surface representation in the top-down pass of our model. Overall this idea of settling surface representation in the top-down pass, and then filling in from the contours to the interior [20, 22] guided our search for the representational choices, and is consistent with the message propagation schedules we use in our model.

The timing and topography of feature activations in the visual cortex has been the inspiration for several models that propose message-passing as a core mechanism for Bayesian inference in cortex [9, 96]. Message-passing in RCN and these models derive from the fact that visual cortex is hierarchically organized, and cortico-cortical connections are abundant and almost always reciprocal [97]. See [9, 98] for reviews of the neuroscience evidence supporting message-passing based Bayesian inference.

8 Supplementary methods and experiments

Here we provide detail-level descriptions of our methodology and the experiments presented in the main paper, as well as supplementary results.

8.1 The preprocessing layer in practice

The PreProc (preprocessing) pipeline converts an image into a set of oriented filter activations. Implementation details of this pipeline are given below.

8.1.1 2D correlation

Given a set of predefined grayscale edge filters, each normalized to have an L_1 norm of 1.0, we first perform a standard 2D correlation on an image with intensity values quantized to integer values 0-255. Typically we use 16 edge orientations where the filters consist of a positive and a negative Gaussian, each with a standard deviation that we refer to as the filter scale. The Gaussians are separated by two standard deviations – an example is shown in Fig. S10. If the input image X is in color, we perform a separate 2D correlation with each color channel, as well as with the grayscale version of the image, and then take the filter scores $F^{\text{corr}} = \textsc{correlation2D}(X)$ to be the element-wise maximum from each of these color and grayscale channels. The output from this process is shown in Fig. S11(b).

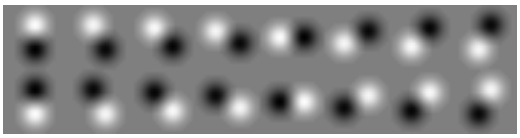
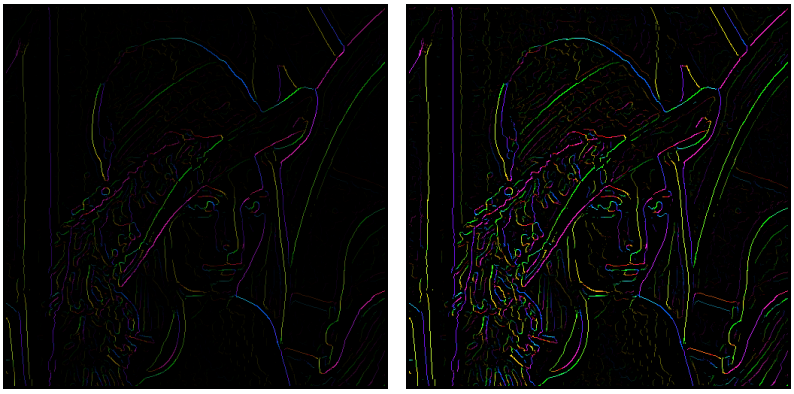


Figure S10: Filters used to detect each of the 16 edge orientations in the input image.

8.1.2 Localization

F^{corr} has the undesirable property that multiple neighboring edge filters will typically be active due to the same edge in the image. To avoid replicating the evidence corresponding to a single edge in the preproc, we localize the those edges with a method similar to that used in Canny edge detection. Fig. S11(c) shows the result of this localization operation. This alleviates the need to perform explaining away between edge filters within the preprocessing layer itself, which empirically we found to be unnecessarily expensive compared to localization, and with no difference in object detection performance.

(a) Input Image (b) 2D Correlation (c) Localized



(d) Cross-Channel Suppression (e) Log Likelihood

Figure S11: Stages of preprocessing. Colors indicate filter orientations, and brightness is proportional to score. Respective values for max brightness threshold, smoothing, and discount parameters (discussed in-text): $\tau = 30$, $\sigma = 0.2$, and $\gamma = 0.9$.

We define the function below to compute $F^{\text{local}} = \textsc{localize}(F^{\text{corr}})$. This function uses the known orientations of the edge filters to perform directional suppression.

$$F^{\text{local}}_{frc} = \begin{cases} 0 \text{ if } F^{\text{corr}}_{frc} < \max_{(\Delta r, \Delta c) \in \textsc{mask}(f)} (F^{\text{corr}}_{f,r+\Delta r,c+\Delta c}) \\ F^{\text{corr}}_{frc} \text{ otherwise} \end{cases} . \tag{S38}$$

$\textsc{mask}(f)$ is an orientation-specific suppression mask that indicates which neighboring filters would be

activated by the same underlying edge in the image. These masks are precomputed to contain the offsets of the filter that are perpendicular to the direction of the edge, and these offsets extend to approximately the width of the filter so that no two filters with the same orientation can occupy the same pixels in the image unless they are aligned in the direction of the underlying edge.

Along with localization, we also perform cross-channel suppression such that at most one filter orientation is active per pixel position. The function to compute these filter scores F^{sup} is defined as

$$F^{\text{sup}}_{frc} = \begin{cases} 0 \text{ if } F^{\text{local}}_{frc} < \max_{f'}(F^{\text{corr}}_{f'rc}) \\ F^{\text{local}}_{frc} \text{ otherwise} \end{cases} . \tag{S39}$$

Fig. S11(d) shows the result of cross-channel suppression.

8.1.3 Conversion to log likelihoods

Next, we normalize F^{pooled} according to a maximum brightness threshold τ:

$$F^{\text{bright}} = \text{CLIP}(F^{\text{sup}}/\tau, 0, 1), \tag{S40}$$

where $\text{CLIP}(X, a, b)$ sets all elements in X less than a to the value of a and greater than b to the value of b. This effectively normalizes the filter scores such that edges with high contrast do not have a proportionally higher score, while edges with very low contrast (where $F^{\text{pooled}} < \tau$) have a brightness-normalized score proportional to the filter score. We typically binarize F^{bright} with all non-1 values set to 0 for captcha images. Then we set F^{loglik} (the log likelihoods of the filter edges being present in the image) as follows:

$$F^{\text{loglik}} = \log(F^{\text{smooth}}) - \log(1 - F^{\text{smooth}}) \tag{S41}$$

$$F^{\text{smooth}} = \sigma + (1 - 2\sigma) \times F^{\text{bright}}, \tag{S42}$$

where $0 < \sigma < 0.5$ is a smoothing parameter that roughly corresponds to the probability that an edge randomly appeared or disappeared due to noise. Fig. S11(e) shows an example of F^{loglik}. In our experiments, we use $\sigma = \frac{1}{1+e}$.

8.1.4 Orientation pooling

We now additionally pool across neighboring filter orientations to compensate for the discretization errors that occur due to cross-channel suppression. When an edge in an image is halfway between two filter orientations, minor noise can cause one orientation to suppress the other, and easily vice versa. To avoid this sensitivity, and to introduce additional invariance to rotation of the object, we amend the preprocessing pipeline to compute F^{pooled},

$$F^{\text{pooled}}_{f,r,c} = \max(F^{\text{loglik}}_{f,r,c}, \max_{f' \in \text{NEIGHBORS}(f) \wedge F^{\text{loglik}}_{f',r,c} > 0} \gamma F^{\text{loglik}}_{f',r,c}), \tag{S43}$$

where $0 < \gamma < 1$ is a discount parameter that reduces the activations of pooled filters that were not maxima during cross-channel suppression. This affords some rotation invariance while still assigning

the highest score to the true edge orientation in the image. NEIGHBORS(f) returns the indices of the two edge filters with adjacent orientations.

While it might seem that orientation pooling is "undoing" the cross-channel suppression, we found empirically that F^{pooled} results in a much more discriminative model than using F^{local} without performing these two additional steps.

This concludes the preprocessing pipeline of the shape model. If desired, multiple filter sizes can be used for more robust object detection. Each filter size independently utilizes this same preprocessing pipeline. There are a few parameters for the above steps, such as the brightness threshold τ. In our experiments, these were set manually when beginning to experiment with a given dataset by looking at a few training images and verifying that contours are complete.

8.2 Filter post-processing in practice

When grayscale filters overlap during shape reconstruction, we need a mechanism to efficiently avoid overcounting the log-likelihood corresponding to overlapping pixels.

Formally, given a set of reconstruction filters F^{bank}, each of which has a known log likelihood from the bottom-up preprocessed image F^{pooled}_{frc}, we wish to correct the aggregate log-likelihood $L_{\text{agg}} = \sum_{(f,r,c)\in F^{\text{bank}}} F^{\text{pooled}}_{frc}$ of these filters to avoid counting the same pixels more than once. We assume that each filter f is defined as a sparse set of (x_i^f, w_i^f) pairs, where vector $x_i^f = (\Delta r_i^f, \Delta c_i^f)$ specifies the offset of a pixel relative to the filter center and w_i^f is the weight of that pixel, such that $\sum_i |w_i^f| = 1 \; \forall f$.

First note that

$$L_{\text{agg}} = \sum_{(f,r,c)\in F^{\text{bank}}} F^{\text{pooled}}_{frc} = \sum_{(f,r,c)\in F^{\text{bank}}} \sum_i |w_i^f| F^{\text{pooled}}_{frc}. \tag{S44}$$

We compute the corrected aggregate log-likelihood L_{recount} as follows:

$$L_{\text{recount}} = \sum_{(f,r,c)\in F^{\text{bank}}} \sum_i \frac{|w_i^f|}{W_{r+\Delta r_i^f, c+\Delta c_i^f}} F^{\text{pooled}}_{frc} \qquad W_{r'c'} = \sum_{(f,r,c)\in F^{\text{bank}}} \sum_i |w_i^f| \delta_{r', r+\Delta r_i^f} \delta_{c', c+\Delta c_i^f} \tag{S45}$$

where δ_{ij} is the standard Kronecker delta. W_{rc} can be interpreted as the sum of the absolute weights of all reconstruction filters at position (r, c), which is used to normalize the relative contribution of each pixel in each filter to the final recount score. Intuitively, when two reconstruction filters share a common image pixel with weights $w_i^{f_1}$ and $w_j^{f_2}$, then the log likelihood from that pixel is reduced in proportion to the sum of the weights $w_i^{f_1} + w_j^{f_2}$. This way, when we sum the reconstruction filter log-likelihoods, the log-likelihood of each pixel is reduced by precisely as much as needed to prevent overcounting.

8.3 Network architectures

Experimenting with different depths for RCN has shown us that for the experiments involving text parsing and individual symbols (CAPTCHA, ICDAR, MNIST, Omniglot), more than two levels does not improve performance – although additional levels did help reduce computation time (see Section 8.11). However, for the more complex setting of parsing scenes that contain realistic renderings of 3D objects, additional levels did increase performance (see Section 8.12 for a comparison between a two-level and a four-level RCN).

Training an RCN network on a given dataset only requires learning the features of the topmost layer for that dataset, since the features of the layers below are found via pretraining on a separate set (the SHREC 3D dataset [75], as described in Section 5), and kept fixed. Different experiments were run with varying parameterizations, which we collect in Table S1. The ICDAR experiment is not included in this table because it does not use a fixed PreProc parameterization or our standard parser. Instead, this particular application uses an adaptive PreProc and a specialized parser, as described in [99].

In all of our experiments we use 16 PreProc channels that capture the presence or absence of edges at 16 different orientations. This results in the variable Y defined in Section 3 being a binary multidimensional array of size $16 \times H \times W$ for images of size $H \times W$. The 16 PreProc channels are depicted in Fig. S10.

Table S1: RCN parameterizations used for each of the experiments. L: number of layers; sc: image scaling before PreProc; fsc: PreProc filter scale; mbt: maximum brightness threshold; ps: pool shape; pf: perturb factor; op: overlap penalty; rwe: pool reweighting exponent; δ: complexity penalty; - : does not apply; range of values: value depends on number of training instances, detailed in the corresponding experiment section; list of values: applied per layer. See Sections 1, 4.7, and 8.1 for further details about the meaning of each of these parameters.

Experiment	Parameter								
		PreProc			Pooling		Parsing		
	L	sc	fsc	mbt	ps	pf	op	rwe	δ
reCAPTCHA	2	2.00	2.0	20	41	2.9	-0.0875	0.5	-0.625
BotDetect	2	1.45	2.0	18	21	3.0	-0.1500	0.700	-1.1
Paypal	2	1.58	1.5	65	15	3.0	-0.1500	0.700	-1.6
Yahoo	2	1.45	2.0	55	27	3.0	-0.1000	0.700	-1.2
MNIST	2	4.00	4.0	40	25 to 57	1.0 to 2.0	-	-	-
Omniglot	2	0.50	3.0	40	47	1.0	-	-	-
Objects	4	0.75	2.0	12	3, 7, 21	2, 3, 8	-	-	-

8.4 CAPTCHA datasets

8.4.1 reCAPTCHA

We downloaded 5500 reCAPTCHA images from the google.com reCAPTCHA page, of which 500 were used as a validation set for parameter tuning, and accuracy numbers are reported on the remaining 5000. Note that since the reCAPTCHA was broken this is no longer allowed. A simple heuristic based on grayscale dilation of each image was sufficient to find the reCAPTCHA word within the image vs the OCR word as the reCAPTCHA had thinner strokes. The Death by Captcha `http://www.deathbycaptcha.com/` service was then used to have reCAPTCHAs hand annotated with their solution. 500 of the reCAPTCHAs formed a training set for tuning the choice of the training font, the PreProc parameters, and the parsing parameters of our system.

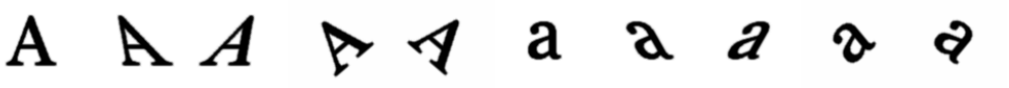


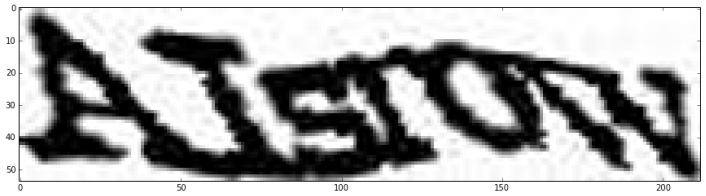
Figure S12: All training examples used for the letter A in the reCAPTCHA network.

The final performance on the test set was that 84.2% of our reCAPTCHA solutions would be accepted by the system. ReCAPTCHAs are accepted if they have have the correct length and at most one character substitution, and the case of the letters is ignored. For context, CAPTCHAs are considered

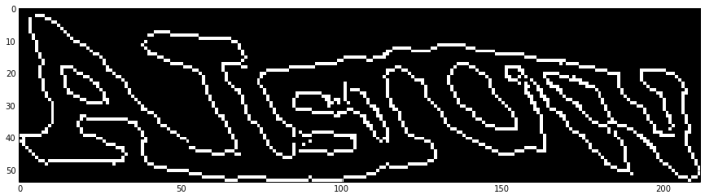
broken if more than 5% of answers are accepted [100]. 66.6% of the reCAPTCHAs had all letters correctly identified, corresponding to a character recognition rate of 94.3% (again ignoring case).

In order to capture smaller details of the reCAPTCHAs using the PreProc, the reCAPTCHAs were scaled up by a factor of two. The PreProc was run with a filter scale of two and a maximum brightness threshold of 20. Due to the nature of distortions present in the reCAPTCHA dataset, edges can vary their orientations. PreProc activations were set to be the max of their activations, 80% of the neighbors' activations (where a neighbor is the next rotated orientation, or 50% of the activations of neighbors once-removed.

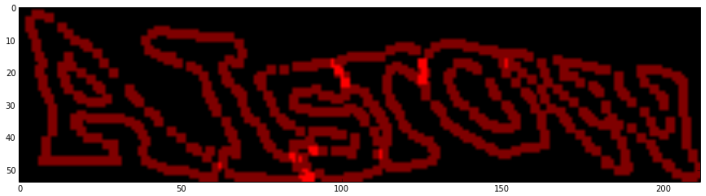
A similar-looking font to those use in reCAPTCHAs, Georgia, was identified by visual comparison from the fonts available on the local system to use in training, as well as a sui table font size. Each image was augmented by either rotations of ±40 degrees or skews of ±40, repeated for upper and lower case letters, to give a total of ten training examples per letter. This resulted in a total of 260 training images, each of which was used to represent a top-level feature in our system. All training examples used for the letter A are illustrated in Fig. S12.



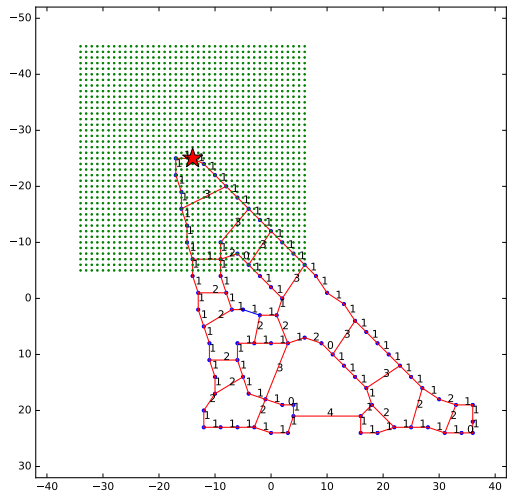
(a) An example reCAPTCHA



(b) Output of PreProc for this image



(c) Winning parse of backtraces



(d) Model derived for one of the training images

Figure S13: (a) An example reCAPTCHA. (b) An example PreProc output, maximization over the direction of the edge filter. (c) The backtraces used to build the winning (and correct) parse. Note that every part of every letter is shown, even if it was not present in the original image, demonstrating the ability of the system to explain the latent causes of the image. (d) The model derived from one of the training examples, with the pool members for a single feature (marked with star) expanded and shown as dots. Each lateral is represented as an edge in the graph, labeled with the maximum allowed relative perturbation in pool choices.

The model was a two-level hierarchy with one pooling layer where we set the pool size to 41, which is sufficient to accommodate the largest amount of global distortion possible according to the laterals. The flexibility of the laterals is controlled via the perturbation factor (pf) defined in 2.3.2, which was set to 2.9. The model derived for one of the training images for the letter A is shown in Fig. S13(d).

The overlap penalty parsing parameter was set to -0.0875 while the pool weighting parameter δ was set to -0.625 by optimizing performance on the training dataset (see Section 4.7). An example parsing

of a reCAPTCHA and PreProc output is shown in Figs. S13(a)-(c).

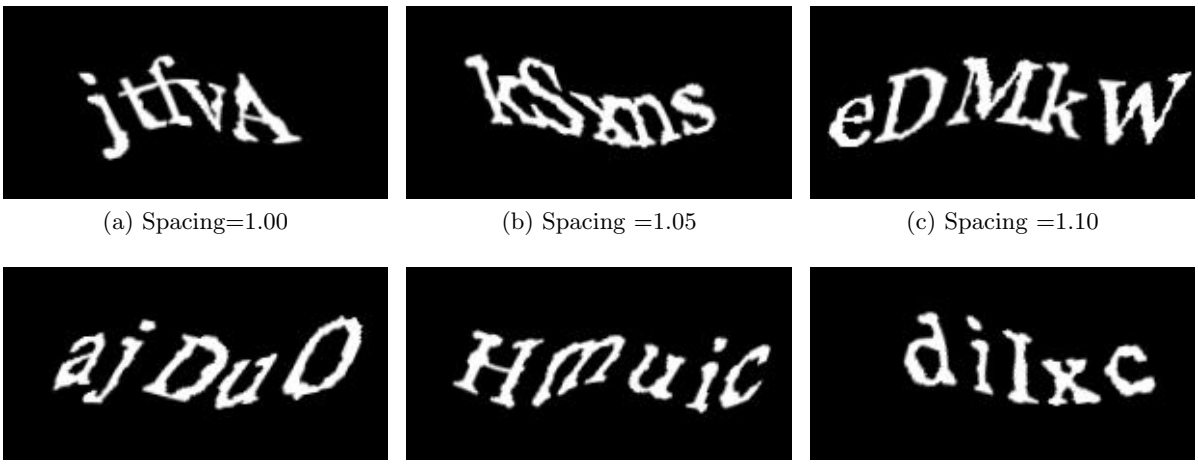
Total training time on a single AWS core of an m4.xlarge instance was 67s. Each of the 5000 reCAPTCHA images took on average 94s on a single core, distributed over an AWS condor cluster where the nodes were of type r3.large. Note that as the inference is highly parallel, a single reCAPTCHA can be parsed within 31s on a 2.7 GHz MacBook Pro Retina 2012 using 7 threads on 4 hyper-threaded cores. Leveraging further CPUs can take that time down to a few seconds per image.

8.4.2 Human accuracy on reCAPTCHA

Human accuracy on reCAPTCHA dataset was estimated with Amazon Mechanical Turk (AMT) using U.S. based workers. AMT workers achieved an accuracy of 87.4%. The accuracy estimate is based on $10,000$ human parses, using the same dataset as RCN's test set. Each test image was parsed by two different AMT workers, and the solutions from the workers agreed only for 81.0% of the reCAPTCHAs.

8.4.3 reCAPTCHA control dataset

As a control experiment, we reproduced the high-level results of a recent convolutional neural network (CNN) based approach [6] that parsed reCAPTCHAs to study its properties. The 5000 labeled examples of reCAPTCHA that we had were not adequate for training this network; the paper reported using millions of labeled training examples. To overcome this problem, we created a CAPTCHA generator that approximately renders reCAPTCHA-like strings. The emulated reCAPTCHA was created using ImageMagick. Each image contains between 5 and 8 characters, with each character taking up the default spacing width, where each individual spacing is multiplied by a uniform random coefficient taken from $[0.82, 1.02)$. The image is subject to a random wave effect with amplitude up to 3, and a random phase shift. It is also subject to a swirl effect with uniformly chosen random swirl coefficient between -16 degrees and +16 degrees. In the testing dataset, the number of characters is restricted to 5. The spacing between characters is multiplied by an additional coefficient, fixed with respect to each image, systematically taken from $\{1, 1.05, 1.1, 1.15, 1.2, 1.25\}$. Examples of generated reCAPTCHAs are shown in Fig. S14.



(a) Spacing=1.00 (b) Spacing =1.05 (c) Spacing =1.10

(d) Spacing =1.20 (e) Spacing =1.25 (f) Spacing =1.30

Figure S14: Examples of generated CAPTCHA images

8.4.4 reCAPTCHA CNN control experiment

The CNN architecture was based on the multi-digit classification network of [6] which used a bank of position-specific networks to classify the characters at each position. We used smaller images, and our architecture makes use of modern insights like residual layers [101] and a preference for deeper nets with smaller convolutional filters.

Table S2: reCAPTCHA CNN Architecture

Layer	Parameters	Activation	Preceded by
conv1	stride=1; conv=5x5; 32 channels	relu	-
conv2	stride=1; conv=3x3; 64 channels	relu	pool 2x2
res3	conv=3x3/3x3; 64 channels	relu	-
res4	conv = 3x3/3x3; 96 channels	relu	pool 2x2
res5	conv = 3x3/3x3; 128 channels	relu	pool 2x2
fc6	512 channels	relu + dropout(p=0.5)	pool 2x2
softmax	8 positions; 53 classes [a-zA-Z$]	-	pool 2x2

We now briefly describe our TensorFlow [102] CNN implementation. The raw image is single-channel, with values ranging from -0.5 to 0.5. Convolutional weights are randomly initialized with mean 0 and standard deviation 0.1. Bias parameters are initialized to 0. We use a momentum optimizer with 0.01 base learning rate, and 0.95 decay rate per epoch. The batch size is 45. Optimization was run for 80 epochs, where data is permuted at the start of every epoch; data was also augmented by random translations of up to 5 pixels in each cardinal direction per epoch. All model parameters use L_2-regularization with coefficient 0.002. Zero-padding is performed under the "SAME" scheme such that the output channel count is preserved for stride 1, and halved for stride 2. When residual layer outputs have more channels than their inputs, the inputs are padded with zeros to match the output channel count. Each residual layer contains a total of two convolution layers.

Our reCAPTCHA CNN architecture is presented in Table S2. The final classification layer used a distinct classifier for each character position as in [6]. The classifiers had 53 categories, consisting of upper and lower case characters, plus a special character indicating positions after the end of the character sequence. We trained on 79,000 images. The validation error on the 1000 images decreased monotonically and approached 0 for each character.

Table S3: reCAPTCHA control dataset performance on spacing different from training

Spacing	1.00	1.05	1.10	1.15	1.20	1.25
CNN Full Parse Accuracy	89.9	80.5	61.6	38.4	14.4	7.0
RCN Full Parse Accuracy	90.4	92.4	93.7	94.3	93.1	93.3

To test the robustness of the system, we systematically varied the spacing between the characters from 1.0 (training spacing) to 1.25 in steps of 0.05 during testing (Fig. S14). In the testing dataset, the number of characters is restricted to 5. The test performance is shown in Table S3. Even though the spaced characters are subjectively much easier to parse, the trained CNN fails on the novel spacing. Uniformly varying the horizontal spacing of the characters is just one of the many perturbations that can be done during testing. In addition, the architecture of the network makes it clear that it cannot handle strings with fewer or more characters than the ones presented during training. Although this issue can be handled directly with data augmentation, the number of combinations involved can make the training infeasible.

8.4.5 RCN on reCAPTCHA control dataset

The RCN network that parsed the original reCAPTCHA strings were able to parse the control dataset with an accuracy rate of 90.4. The accuracy of the network was robust to perturbations in spacing of the characters. Overall, the accuracy improved with the increase in spacing.

Unlike the CNN described above, RCN learns generative models for individual characters that are then used for parsing. In RCN, the interactions between characters are resolved during parsing, making it robust to variations in spacing, overlap, number of characters in the string, and similar variations. The CNN approach requires these interactions to be present during training, increasing its training complexity, and making it brittle to perturbations that are not present during training.

8.4.6 BotDetect

BotDetect is a CAPTCHA system used widely on the Internet that offers many styles of CAPTCHAs designed to be difficult to parse by computers. As many of the CAPTCHA styles used similar techniques to make the CAPTCHA hard to break, a subset of the styles were selected that illustrated many of these diverse confounding effects. Variations in the CAPTCHAs included adding clutter, missing edges, inconsistent fill patterns, and background clutter. See Fig. S15 for an example from each of the CAPTCHA styles we chose, the corresponding PreProc output, and the final per character and parsing performance. CAPTCHAs were downloaded using the demo page from their website.

Death By Captcha was used twice to label each image and any disagreements between the labelings were resolved by hand. We downloaded a dataset of 50-100 images per CAPTCHA style for determining the parsing parameters and training setup and another 100 images as a testing dataset on which the network is not tuned.

As training images for the system, we selected by visual inspection, a series of fonts and scales from those available on the system by looking at a few examples of the BotDetect CAPTCHAs. These were fonts from the standard OS X font library, and were Arial, Arial Bold, Menlo, Georgia, Georgia Bold, and Courier New Bold; all were used at font size 75, apart from Courier New Bold that used font size 90. Only characters that were observed to occur in the CAPTCHAs were included, which were "3456789ABCDEHJKMNOPRSTUVWXYZ". Each training image was also presented as unscaled or rescaled by a factor 1.2, and rotated either -10 or +10 degrees. This gives a total of 24 training images per case per character. The BotDetect test images were rescaled by a factor of 1.45.

The PreProc used a filter scale of 2 and a maximum brightness of 18. As in the reCAPTCHA experiment, due to the nature of distortions present in the dataset, edges can vary their orientations. We set PreProc activations to be the max of their activations, the activation of the opposite orientation, 95% of the neighbors' activations (where a neighbor is the next rotated orientation), 60% of the activations of neighbors once-removed, or 50% of the opposite orientation's neighbors. This was done as various CAPTCHAs would transition between light-to-dark and dark-to-light edges, as demonstrated by the chess board dataset seen in Fig. S15.

This was a two-level hierarchy with one pooling layer with pool size 21. The flexibility of the laterals is controlled via pf (see Section 2.3.2), which was set to 3.

Parsing was performed with an overlap penalty of -0.15, δ set to -1.1, and the exponent of the pool re-weighting was set to 0.7 instead of the square root directly being used (see Section 4.7 for details). We also performed experiments to test the transferability of parameter settings between datasets in the next section.

Note that this model can be applied to captchas of arbitrary image size with arbitrary number of letters present. This is in contrast to the convolutional neural network (CNN) based approach to solving captchas (as done for example by [6]). This required training on both a specific image size and

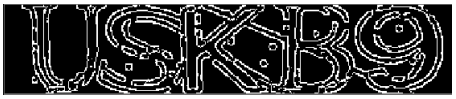
Dataset	Image	PreProc	$C_{\%}$	$F_{\%}$
Fingerprints			97.0	86
Jail			66.5	13
Ghostly			87.4	51
Mosaic			95.2	78
Bullets			91.2	63
Melting heat			96.3	83
Snow			90.0	59
Halo			93.9	73
Chess			92.6	68
Wave			84.9	44
Average			90.8	61.8

Figure S15: Example BotDetect images and their corresponding PreProcs, demonstrating the wide variety of different CAPTCHAs that can be broken with the same network. $F_{\%}$ is the full parse accuracy, and $C_{\%}$ is an estimate of the per character accuracy obtained as $F_{\%}^{0.2}$.

on the maximum number of characters present. The fact that there are always five letters present in the bot detect captcha is could be exploited for additional parsing performance.

8.4.7 BotDetect with the appearance model

We test a simple special case of the appearance model described in Section 3 where $\alpha = 0$. This allows no underlying color variation for pixels contained within the mask, and a single underlying grayscale color $h \in [0, 1]$ is assumed for all r, c where $Y_{r,c}$ is in the IN state. For those positions, the observed color at a pixel $X_{r,c}$ is a mixture of two uniform distributions. It is with probability m a sample from $U[h - \gamma, h + \gamma]$ or with probability $1 - m$ sampled uniformly $U[0, 1]$, where $U[a, b]$ is a uniform distribution between a and b. All positions not in the mask, where $Y_{r,c}$ is in the OUT state, are assumed to be sampled randomly from $U[0, 1]$ independent of the value of $Y_{r,c}$.

From these, we can derive the following log-odds for every pixel in the mask (i.e., where $Y_{r,c}$ is in the IN state):

$$\text{LogOdds}_{r,c} = \log P(X_{r,c}|Y_{r,c} = h \text{ is in state IN}) - \log P(X_{r,c}|Y_{r,c} \text{ is in state OUT}). \tag{S46}$$

The log-odds of a pixel not in the mask is always 0. Empirically we found that instead of directly using this formulation it is better to replace $LO_{r,c}$ with a formulation that incorporates the original edge score as follows:

$$\text{PixelScore}_{r,c} = \text{EdgeScore}_{r,c} + \begin{cases} 0, & \text{if } Y_{r,c} \text{ is in state OUT} \\ \psi, & \text{if } Y_{r,c} \text{ is in state IN and } X_{r,c} - h \leq \gamma \\ \chi, & \text{if } Y_{r,c} \text{ is in state IN and } X_{r,c} - h > \gamma \end{cases}. \tag{S47}$$

ψ and χ are more flexible than the log odds scores derived from m and γ as they also contain an implicit normalization of the score compared to the EdgeScore. In practice h is extremely peaked for each mask so the maximum value of h is used instead of marginalization. This gives a final per image score:

$$\text{ImageScore} = \max_h \sum_{r,c} \text{PixelScore}_{r,c}(h). \tag{S48}$$

Each of the top N parses produced by the parsing algorithm using the edge scores was re-scored by incorporating the appearance model as described above. For the purposes of these experiments, we assumed that $\alpha = 0$ – i.e., a single unified color was assumed to underlie every parse. A parameter γ was defined to be the maximum deviation allowed from the underlying color h.

We optimized the parameters on a training set of 100 images for each of the 10 datasets, giving 1000 images in total. Parameters that were optimized were the log likelihood bonus ψ for having a mask pixel be within γ of the underlying color and the penalty χ for having a mask pixel not be within γ of the underlying color. We searched over the following sets of values for each parameter:

$$\begin{aligned} \psi &\in [0.001, 0.0025, 0.005, 0.01, 0.025, 0.05] \\ \chi &\in [-0.001, -0.0025, -0.005, -0.01, -0.025, -0.05] \\ \gamma &\in [2, 4, 8, 16, 32, 64, 128] \end{aligned}$$

The best combination on the training set was found to be $\gamma = 32, \psi = 0.01, \chi = -0.01$. A single set of parameters was used for all datasets.

We then evaluated these parameters on an unseen test set of 1000 images. We found these parameters on unseen testing dataset resulted in 64.4% full parse accuracy compared to the edge test performance of 61.8%. This demonstrates that incorporating the appearance model leads to a performance increase of around 2.6% on the 1000 novel test images.

8.4.8 Determining the transferability of the BotDetect parsing parameters

Parsing parameters were optimized in turn on each of the 10 unseen test datasets, and the effect of those parameters was evaluated on the other nine datasets. Each dataset has 100 images.

Overlap penalties tested were $[0.1, 0.25, 0.5, 1, 2]$, while the δ values were $[-0.1, -0.25, -0.5, -1, -2]$, and the exponents of the pool re-weighting were $[0.5, 1, 2]$. All combinations were evaluated on training datasets and the one that performed best on the training dataset was evaluated on the other nine datasets.

The final training performance was $41 \pm 26\%$ while the validation performance was $48 \pm 8\%$. The high training set variability comes from each dataset being of varying difficulty, as illustrated in Fig. S15. The similar training and test set mean performance and low variability of the test set performance demonstrates that the parsing parameters are not over optimized for the specific dataset and the parameters transferred well to the other datasets.

8.4.9 PayPal

We downloaded 112 CAPTCHAs by hand from the PayPal website via repeated entry of incorrect passwords and labeled them by hand. Using a similar network as the BotDetect and reCAPTCHA networks we achieved a performance of 57.1% full parse accuracy.

The test images were rescaled by a factor of 1.58, the PreProc used a filter scale of 1.5 and a maximum brightness of 65. As in BotDetect, due to the nature of distortions present in the dataset, edges can vary their orientations. PreProc activations were set to be the max of their activations, 95% of the neighbors' activations (where a neighbor is the next rotated orientation), 60% of the activations of neighbors once-removed. This is the same as the BotDetect settings apart from the lack of pooling over opposite orientations.

Visual inspection was used to identify two similar system fonts, Arial and LucidaGrande at font sizes 48, 58, 60 and 72.5, for a total of eight training images per character per case. Only letters and digits that occur in the PayPal CAPTCHA were included, which were "23456789ABCDEFGHJKLMNPRSTUVWXYZ". PayPal CAPTCHAs include clutter created by the letters of the PayPal logo, but the CAPTCHA letters to be decoded were of a larger size, and model-based segmentation was able to filter out the clutter without any special pre-processing.

The pooling size was set to 17 and the perturbation factor was set to $\text{pf} = 3$. Parsing was performed with an overlap penalty of -0.15, δ set to -1.6, and the exponent of the pool re-weighting was set to 0.7 instead of the square root directly being used. See Section 4.7 for details.

8.4.10 Yahoo

We downloaded 87 CAPTCHAs by hand from the Yahoo website via repeated entry of incorrect passwords and labeled them by hand. Using a very similar network to the BotDetect and reCAPTCHA networks we achieved a performance of 57.4% full parse accuracy.

The test images images were rescaled by a factor of 1.45, the PreProc used a filter scale of 2 and a maximum brightness of 55. Due to the nature of distortions present in the dataset, edges can vary their orientations. Pooling over neighboring activations was identical to that used in PayPal.

Visual inspection was used to identify similar system fonts: Arial, Book Antiqua Bold, Bookman Old Style, Courier New Bold, and Times New Roman at font size 123. We additionally included bold, and bold and italic versions of Arial and Times New Roman to gave a total of 10 training images per letter per case. Only letters and digits that occur in Yahoo were used which were the digits "2345678", the letters "ABFGHJLMPTV" in upper case and "bcdenprsuwyz" in lower case.

The pooling size was set to 27 and the perturbation factor was set to pf = 3. Parsing was performed with an overlap penalty of -0.1, δ set to -1.2, and the exponent of the pool re-weighting was set to 0.7 instead of the square root directly being used – see Section 4.7 for more details.

8.4.11 Using the same font set for parsing different CAPTCHAs

The experiments above used different sets of fonts for different families of CAPTCHAs. We have done a limited set of experiments that train a single model on a larger variety of fonts to parse different CAPTCHA types. Although the font models can be generic and transferred between CAPTCHA types, the parsing parameters need to be set separately for optimal accuracy. Currently, the background model is not dynamically estimated from each image. Moreover, the parser used for CAPTCHAs does not have any prior on the layout of characters in a scene. Experiments described in the next section demonstrate how a single character model trained on a variety of fonts can parse text in uncontrolled environments, when the character model is combined with a scene parser that exploits rich prior knowledge about scene layouts.

8.5 ICDAR: text recognition in uncontrolled environments

ICDAR ("International Conference on Document Analysis and Recognition") is a biannual competition on text recognition. We tested RCN on ICDAR 2013 Robust Reading Competition in which the objective is developing text recognition methods for real world images [54]. For this test, we enhanced the parsing algorithm to include prior knowledge about n-grams and word statistics, and about geometric priors related to the layout of letters in a scene, which includes spacing, relative sizes, and appearance consistency. We compared our result against top participants of the ICDAR competition, and against a recent deep learning approach. Compared to born-digital images like those used on the web, real world image recognition is more challenging due to uncontrolled environments and imaging conditions that results in strong variation in font, high blur/noise, various distortions, non-uniform appearance, and strong background structure. In this experiment, we only consider letters, ignoring punctuations and digits. All test images are cropped and each image contains only a single word – see examples in Fig. S16.

8.5.1 Methods

Our text recognition pipeline is two-fold. First, letter candidates are detected using RCN. Second, a parsing factor graph is constructed to infer the true word from the many letter candidates.

Step 1: Character detection
We use RCN to detect letter candidates from the entire image. One important concern is to make the network cope with the huge variation of fonts in uncontrolled, real world images. To ensure sufficient coverage, we obtained 492 fonts from Google Fonts[17]. However, it is not feasible to use all fonts for character detection (492 fonts times 52 letters gives 25584 training images), and manual font selection is biased and inaccurate.

We resort to an automated greedy font selection approach. Briefly, we render binary images for all fonts and then use the resulting images of the same letter to train an RCN. This RCN is then used to

[17] `https://www.google.com/fonts`

Figure S16: Selected examples of text in real world images from the ICDAR 2013 dataset. Strong variation in font, background, appearance, and distortion makes recognition difficult.

recognize the exact images it was trained on, providing a compatibility score (between 0.0 and 1.0) for all pairs of fonts of the same letter. Finally, using a threshold (=0.8) as the stopping criterion, we greedily select the most representative fonts until 90% of all fonts are represented.

After font selection for all letters we retained 776 unique training images in total (equivalent to a compression rate of 3% w.r.t. training on all fonts for all letters). Fig. S17 shows the selected fonts for letter "a" and "A", respectively.

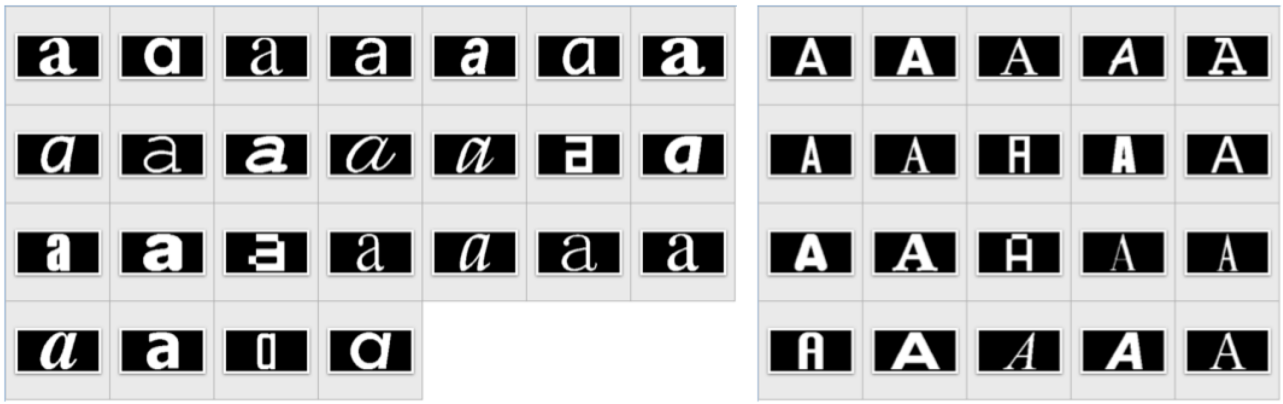


Figure S17: Greedy font selection results for letter "a" and "A" from 492 fonts.

Step 2: Word parsing
RCN was shown to provide high invariance and sensitivity, yielding a rich list of candidate letters that contains many false positives. For example, an image of letter "E" may also trigger the following: "F", "I", "L", and "c". Word parsing refers to inferring the true word from this rich list of candidate letters.

Our parsing model can be represented as a high-order factor graph (see Fig. S18). First, we build hypothetical edges between a candidate letter and every candidate letter on its right-hand side, subjective to some distance range. Two pseudo letters "*" and "#" are created, indicating *start* and *end* of the graph, respectively. Edges are created from *start* to all possible head letters and similarly

from *end* to all possible tail letters. Each edge is considered as a binary random variable which, if activated, indicates a pair of neighboring letters from the true word.

We define four types of factors. Transition factors (green, unary) describe the likelihood of a hypothetical pair of neighboring letters being true. Similarly, smoothness factors (blue, pairwise) describe the likelihood of a triplet of consecutive letters. Two additional factors are added as constraints to ensure valid output. First, consistency factors (red, high-order) ensure that if any candidate letter has an activated inward edge, it must have one activated outward edge. This is sometimes referred to as "flow consistency". Second, to satisfy the single-word constraint, a singleton factor (purple, high-order) is added such that there must be a single activated edge from "start". Examples of these factors are shown in Fig. S18.

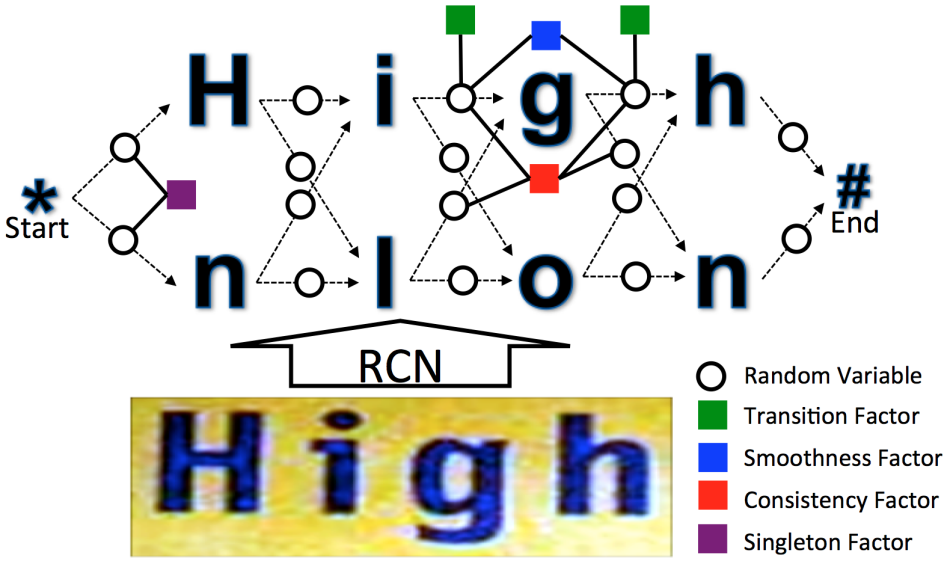


Figure S18: Example of our parsing factor graph. Given an image, RCN generates a list of candidate letters. A factor graph is created by creating edges between hypothetical neighboring letters and considering these edges as random variables. Four types of factors are defined: transition, smoothness, consistency, and singleton.

Mathematically, assuming potentials on the factors are provided, inferring the state of random variables in the parsing factor graph is equivalent to solving the following optimization problem, which can be done efficiently using a second-order Viterbi algorithm.

$$
\begin{aligned}
\hat{\boldsymbol{z}} = \quad & \arg\max_{\boldsymbol{z}} \quad \left\{ F(\boldsymbol{z}; \boldsymbol{w}^{\mathrm{T}}, \boldsymbol{w}^{\mathrm{S}}) := \sum_{c \in \boldsymbol{C}} \sum_{v \in \mathrm{Out}(c)} \phi_v^{\mathrm{T}}(\boldsymbol{w}^{\mathrm{T}}) z_v + \sum_{c \in \boldsymbol{C}} \sum_{\substack{u \in \mathrm{In}(c) \\ v \in \mathrm{Out}(c)}} \phi_{u,v}^{\mathrm{S}}(\boldsymbol{w}^{\mathrm{S}}) z_u z_v \right\} \\
& \text{s.t.} \quad \forall c \in \boldsymbol{C}, \sum_{u \in \mathrm{In}(c)} z_u = \sum_{v \in \mathrm{Out}(c)} z_v, \\
& \qquad \sum_{\forall v \in \mathrm{Out}(*)} z_v = 1, \\
& \qquad \forall c \in \boldsymbol{C}, \forall v \in \mathrm{Out}(c), z_v \in \{0, 1\},
\end{aligned} \tag{S49}
$$

where,

- $\boldsymbol{z} = \{z_v\}$ is the set of all binary random variables indexed by v;
- $\boldsymbol{C}$ is the set of all candidate letters, and for candidate letter c in $\boldsymbol{C}$, $\mathrm{In}(c)$ and $\mathrm{Out}(c)$ index the random variables that correspond to the inward and outward edges of c, respectively;

- $\phi_v^{\mathrm{T}}(\boldsymbol{w}^{\mathrm{T}})$ is the potential of transition factor at v (parameterized by weight vector $\boldsymbol{w}^{\mathrm{T}}$), and $\phi_{u,v}^{\mathrm{S}}(\boldsymbol{w}^{\mathrm{S}})$ is the potential of smoothness factor at u and v (parameterized by weight vector $\boldsymbol{w}^{\mathrm{S}}$);
- The three constraints ensure flow consistency, singleton and all random variables' binary nature.

Another issue is proper parameterization of the factor potentials, i.e. $\phi_v^{\mathrm{T}}(\boldsymbol{w})$ and $\phi_{u,v}^{\mathrm{S}}(\boldsymbol{w})$. Due to the complex nature of real world images, high dimensional parsing features are required (20 in this case). For example, consecutive letters of the true word are usually evenly spaced. Also, a character-gram model can be used to resolve ambiguity and significantly improve the parsing quality. We use Wikipedia as the source for building our character-gram model. Both $\phi_v^{\mathrm{T}}(\boldsymbol{w})$ and $\phi_{u,v}^{\mathrm{S}}(\boldsymbol{w})$ are linear models of some features and a weight vector $\boldsymbol{w}$. To learn the best weight vector that directly maps the input-output dependency of the parsing factor graph, we used the maximum-margin structured output learning paradigm [103].

Lastly, top scoring words from the second-order Viterbi algorithm are re-ranked using word-level features, such as frequency in Wikipedia.

8.5.2 Results and comparison

We compared our result against top participants of the ICDAR 2013 Robust Reading Competition[18]. Results are given in Table S4. Our model perform better than PhotoOCR [104] by 1.9%. However, the more important message is that we achieved this result using three orders of magnitude less training data – 1406 total images (776 font images for training RCN and 630 word images for training the parser) vs 7.9 million by PhotoOCR. Two major factors contribute to our high data efficiency. First, considering character detection, RCN directly models shape and invariance, showing very strong generalization in practice. Data-intensive models like those in PhotoOCR impose no structure or prior, and require significantly more supervision. Second, considering word parsing, RCN solves recognition and segmentation together, allowing the use of highly accurate parsing features. On the other hand, PhotoOCR's neural-network based character classifier is incapable of generating accurate boundaries for the letter, instead providing only a classification label, making the parsing quality bounded.

Table S4: Performance comparison on the ICDAR 2013 Robust Reading Competition.

Method	Accuracy	Total No. of Training Images
PLT (OmniPage OCR) [54]	64.6%	N/A
NESP [54]	63.7%	N/A
PicRead [54]	63.1%	N/A
Deep Structured Output Learning [55]	81.8%	8,000,000
PhotoOCR [104]	84.3%	7,900,000
Ours	**86.2%**	**1,406**

Fig. S19 shows some typical failure cases for our system (left) and PhotoOCR (right). Our system fails mostly when the image is severely corrupted by noise, blur, or over-exposure, and when the text is handwritten. PhotoOCR fails randomly on many clean images where the text is easily readable. This reflects the limited generalization of data-intensive models.

8.6 One-shot classification and generation for Omniglot dataset

We applied RCN to one-shot learning on the Omniglot handwriting dataset[19]. We consider two one-shot learning tasks: (1) one-shot classification among within-alphabet characters, and (2) one-shot generation of new examples.

[18] `http://dagdata.cvc.uab.es/icdar2013competition/`
[19] `http://www.omniglot.com/`.

Figure S19: Examples of failure cases for our system and PhotoOCR. Typically our system fails when the image is severely corrupted or contains handwriting. PhotoOCR is prone to failing at clean images where the text is easily readable.

For one-shot classification, we use the same training and test dataset[20] as in [1]. The dataset has 20 alphabets, each consisting of 20 different characters with one training and one test example per character. A 20-way classification is performed for each alphabet, and the average accuracy is reported. We trained a two-level RCN system with the same features used in the MNIST experiments. The images are downsampled by a factor of 0.5, and the pooling size and perturbation factor are set to be 47 and 1.0, respectively, both chosen using a separate validation dataset[21]. The average classification error rate for RCN is 7.25%, which is worse than the Bayesian program learner (BPL)'s 3.3% [1], but is better than a deep convolutional network (13.5%) and a deep Siamese convolutional network that optimized for this one-shot learning task (8.0%).

We do not expect RCN to outperform BPL on this task because BPL has the advantage of modeling the causal writing process. However, the BPL method crucially depends on extracting the skeleton information of the strokes during inference; this would fail if the image is cluttered or noisy. RCN is good at segmenting the strokes out from clutter, on top of which a BPL-like mechanism could be run. It might be possible to make the "parallel" RCN mechanism and the "serial" BPL mechanism to be combined in the setting of probabilistic programming [105, 106].

Fig. S20 shows one-shot generated examples using RCN from four different Omniglot alphabets. In each subfigure, the first row shows the training examples and the subsequent rows show five different samples generated by RCN from that single training example. The samples demonstrate that RCN is able to generate variations of each character while still retaining its identity.

8.7 Classification of MNIST dataset and its noisy variants

MNIST is a well known and widely studied public dataset of handwritten digits (0 to 9). We evaluate RCN models on both the standard noiseless MNIST dataset and its noisy variants, and compare the performance against CNNs. We implement a version of the RCN model with two feature layers and one single translational pooling layer (called two-level RCN). The pooling layer contains two parameters, the pool size and perturbation factor, where smaller perturbation factors mean more flexible configurations between two adjacent pools. We up-sample the original image by a factor of 4 to reveal more details during the sparsification.

[20] Downloaded from `https://github.com/brendenlake/omniglot`.

[21] Both 20- and 5-alphabet *background* validation datasets in [1] lead to almost identical parameter choice and testing performance for RCN.

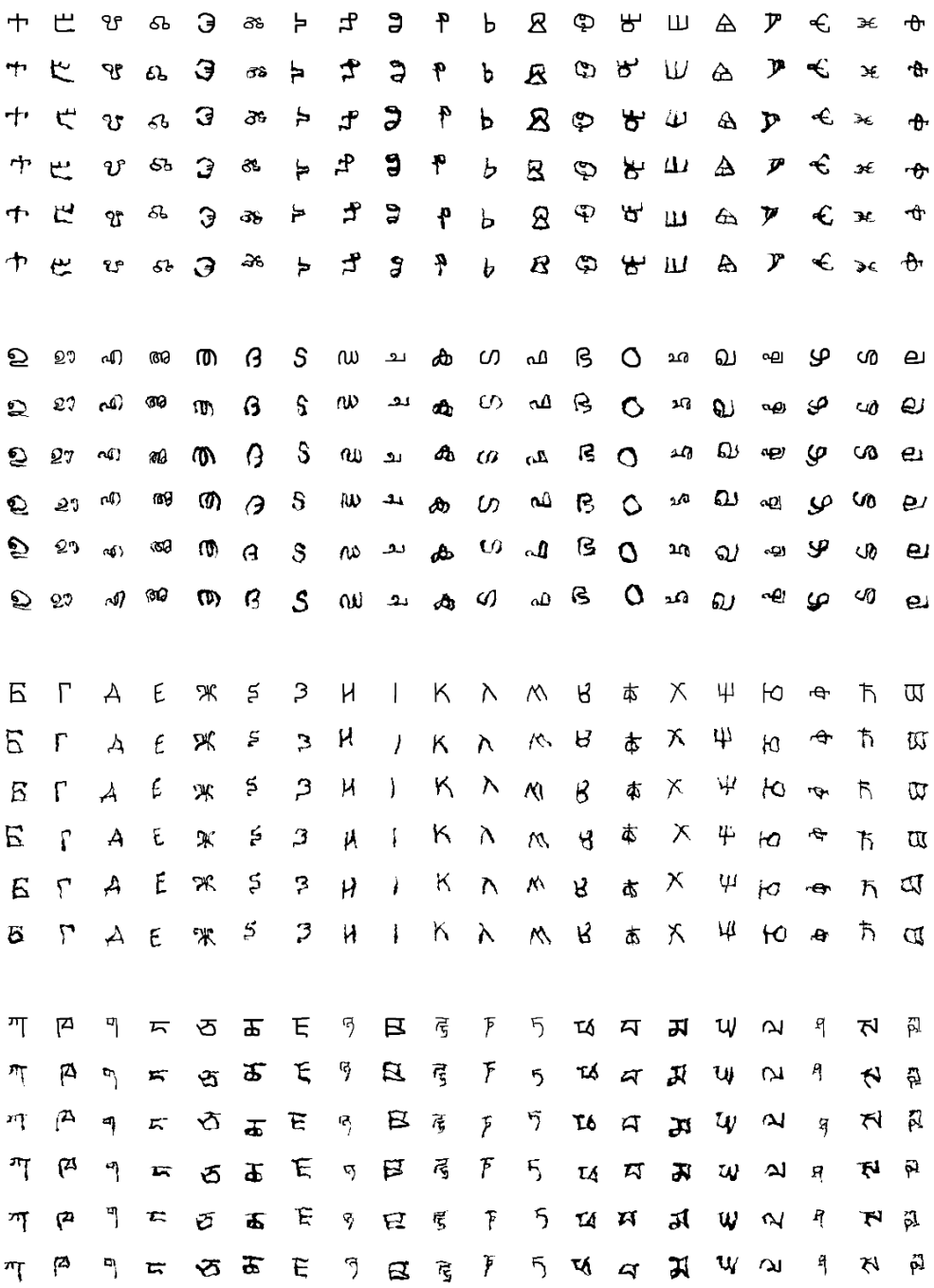


Figure S20: One-shot samples from Omniglot for four different datasets. In each set, the first row shows the training examples and subsequent rows show five different samples from each training example.

8.7.1 CNN control experiments

One family of the CNN baselines was to fine-tune a pre-trained VGG16 model [46]. We tried training a softmax MNIST classifier on top of the fc6, fc7, or fc8 activations respectively. In this experiment, the white-on-black input images were transformed to be in [0, 255] and sized to be of equal dimensions before applying an image-wide offset specific to the published VGG16 model. For each training size, a logistic regression classifier with negligible L_2 regularization was then trained on the internal fc6, fc7, or fc8 activations, leading to three control experiments. This process was done 50 times (for training sizes one and five per digit), and 10 times (for larger training sizes) to establish a mean and standard deviation for accuracy performance. We show only the performance for VGG-fc6 since it is the strongest out of the three.

In another control experiment, we also trained on a network loosely based on the MNIST Tensorflow [102] tutorial, which is based on LeNet-5 [45]. The differences from the original LeNet-5 are as follows: padding is done using the "SAME" scheme, thereby leading to a 7×7 shape for pool2; we use L_2-regularization on all weights and biases with strength 5×10^{-4}; all convolutional weights are initialized using a truncated normal with standard deviation 0.1 and mean 0. As was the case for the VGG control experiments, this process was repeated 50 times (for training sizes 1 and 5 per digit), and 10 times (for larger training sizes) to establish a mean and standard deviation for accuracy performance.

8.7.2 Classification of noiseless MNIST with low training complexity

The standard MNIST dataset contains roughly 60,000 training and 10,000 testing images for digits 0 to 9. We divide the original training dataset into two parts: the first 5000 images for each digit are used as training examples for learning the RCN, and the remaining 1000 images for each digit are used to create a validation set for choosing hyper-parameters. We focus on the low-sample regime performance; in particular we consider the classification performances when the networks are trained (or fine-tuned in the case of CNN) with 1, 5, 10, 20, 30, 50, and 100 examples per digit. In most training settings, we randomly sample 10 sets of training examples and report the averaged accuracy. For the two settings where we use only one and five training examples, we sample 50 randomized trials to address the sampling variance.

Among the different CNN architectures and pre-training settings we compared, the fc6 layer of the VGG network pre-trained on ImageNet gave the best performance. The VGG network (up to fc6) was pre-trained on ImageNet, and the single fully connected layer between fc6 and softmax was tuned for MNIST. We investigated several other CNN architectures, some of which included pre-training on SHREC data rendered as contours only as done for training the features of RCN. The contours-only setting was detrimental to performance of CNNs in all the configurations we tried (Table S5). The LeNet baselines were evaluated under several settings: (1) with pre-training on SHREC-masks, (2) with no pre-training, and (3) with 16 channels of contour features as input. During fine tuning, the weights of the two convolution layers were fixed, but the weights of the two fully connected layers were adapted. Across all the architectures, pre-training using contour pre-processing resulted in decreased performance. The LeNet with SHREC-mask pre-training performed better than the one without pre-training at low data settings, but not better than VGG-fc6. We also performed a further set of experiments (labeled as CNN in Table S5) where we modified the LeNet CNN such that its input was the 16-channel output of the PreProc features used for RCN. However, these configurations did not improve the one-shot or few-shot accuracy of the CNN.

Table S5 shows the classification accuracies of RCN and the other methods on the original MNIST test dataset, and Fig. S21 gives a visual comparison. We also include comparison with the compositional patch model (CPM) of [48], designed for one-shot learning; results are directly from the paper. For each training setting, we also report the two hyper-parameters that were chosen from an independent validation set for the RCN. An independent experiment on rotated MNIST digits was used to deter-

mine the *two* pooling hyper-parameters based on the amount of available training data. We create a new dataset by rotating the MNIST validation set by 90°, and then find, for each required training set size (1, 5, 10, 20, 30, 50, 100), the best two hyper-parameters via cross-validation, using a coarse grid search. The sensitivity of RCN to these two hyper-parameters is not high. Even with a single parameterization, RCN still outperformed CNNs and the CPM in one shot and few-shot classification, but with reduced accuracy in relation to the setting where the hyper-parameters where adapted to the training set size. We show this in Table S6 for a fixed parameter setting.

Table S5: Classification results on noiseless MNIST data. Standard deviations are reported in parenthesis.

# training examples per class	1	5	10	20	30	50	100
Pool size	57	45	45	37	37	33	25
Perturbation factor	1.0	1.0	2.0	2.0	2.0	2.0	2.0
RCN (%)	**76.59**	**92.04**	**94.99**	**96.47**	**97.28**	**97.54**	**97.89**
	(5.66)	(1.30)	(0.72)	(0.29)	(0.18)	(0.20)	(0.16)
VGG (%)	54.18	84.02	90.77	94.65	95.80	96.87	97.77
	(4.96)	(2.20)	(0.86)	(0.56)	(0.82)	(0.43)	(0.24)
LeNet-SHREC-Mask (%)	51.02	79.01	87.47	90.69	92.64	94.17	95.47
	(4.56)	(2.33)	(1.09)	(1.13)	(0.62)	(0.84)	(0.48)
LeNet (%)	46.88	75.15	86.39	91.56	94.28	95.38	96.84
	(4.85)	(2.94)	(2.19)	(0.85)	(0.91)	(0.91)	(0.36)
LeNet-Contours (%)	39.80	65.74	72.00	75.13	74.28	76.33	77.98
	(4.17)	(2.97)	(4.71)	(4.02)	(4.36)	(4.69)	(6.04)
CNN-Contours (%)	38.08	65.05	75.04	79.82	83.42	85.63	88.20
	(3.71)	(3.28)	(1.66)	(0.78)	(1.54)	(1.41)	(0.77)
CNN-SHREC-Contours (%)	28.52	52.10	65.56	74.72	78.86	82.28	86.42
	(3.21)	(3.08)	(1.73)	(1.32)	(2.51)	(1.63)	(0.81)
CPM (%)	68.86	83.79	~ 87	N/A	N/A	N/A	N/A

Table S6: RCN classification on noiseless MNIST data for adaptive vs fixed parameterizations

# training examples per class	1	5	10	20	30	50	100
pool size, perturbation factor	57, 1.0	45, 1.0	45, 2.0	37, 2.0	37, 2.0	33, 2.0	25, 2.0
RCN adaptive (%)	76.59	92.04	94.99	96.47	97.28	97.54	97.89
pool size, perturbation factor	45, 2.0	45, 2.0	45, 2.0	45, 2.0	45, 2.0	45, 2.0	45, 2.0
RCN fixed (%)	70.71	91.38	94.99	96.48	97.15	97.21	97.28
VGG (%)	54.18	84.02	90.77	94.65	95.80	96.87	97.77

In the low-sample regime RCN outperforms both CNN and CPM. The difference is particularly stark in the one- and five-shot learning cases. It is interesting to note that the found pooling hyper-parameters (pool size and perturbation factor) are, as expected, allowing for less and less distortion as the number of templates of each category increases; i.e., the perturb factor is inversely proportional to the allowed distortion, hence it increases.

When trained on the full MNIST training set, RCN achieves 99.24% classification accuracy. Although slightly worse than our CNN baseline (99.40%), it compares favorably with other generative models, e.g., convolutional deep belief networks (with SVM) [107] (99.18%).

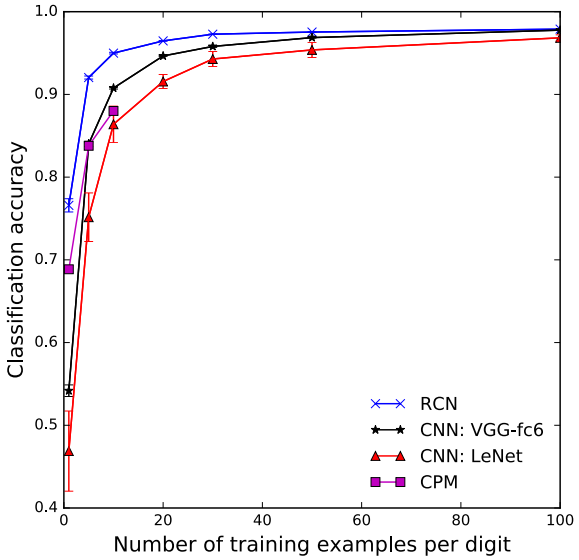


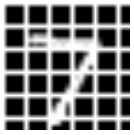
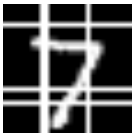
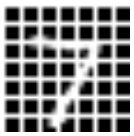
Figure S21: Low-sample classification results on MNIST (Mean ± SEM).

8.7.3 Classification of noisy variants of MNIST

Next we test the robustness of the RCN model on noisy variants of the MNIST datasets. We trained RCN models with 100 training examples per digit (*RCN-1k*) using the same parameters reported in Table S5. As comparison baselines, we consider two variants of the CNN described above, one fine-tuned using 100 training examples per digit (*CNN-1k*), the other using the full training dataset (*CNN-60k*). We use the same 10 sets of randomly selected training examples as before, and report the average classification accuracies over 10 runs in Table S7.

The noisy MNIST dataset is generated from all 10,000 MNIST test images, with six varieties of noise added: background noise, border, patches, grid, clutter, and deletion. Each type of noise has three levels of intensity. Examples of the noisy MNIST test images and the average classification accuracies are shown in Table S7 (numbers in brackets are standard deviations).

Table S7: Classification results on noisy MNIST data. Standard deviations are reported in parenthesis.

Noise Type	noise	border	patches	grid	clutter	deletion
Noise Level 1						
CNN-1k	0.495	0.713	0.864	0.100	0.212	0.756
	(0.066)	(0.055)	(0.018)	(0.016)	(0.027)	(0.019)
CNN-60k	0.915	0.961	0.961	0.239	0.432	0.905
RCN-1k	**0.970**	**0.978**	**0.966**	**0.641**	**0.818**	**0.948**
	(0.002)	(0.001)	(0.001)	(0.018)	(0.005)	(0.002)
Noise Level 2						
CNN-1k	0.331	0.582	0.788	0.096	0.166	0.669
	(0.058)	(0.090)	(0.030)	(0.019)	(0.018)	(0.032)
CNN-60k	0.822	0.890	0.931	0.171	0.331	0.844
RCN-1k	**0.959**	**0.971**	**0.955**	**0.583**	**0.771**	**0.935**
	(0.004)	(0.001)	(0.002)	(0.014)	(0.007)	(0.003)
Noise Level 3						
CNN-1k	0.186	0.417	0.711	0.106	0.141	0.591
	(0.044)	(0.085)	(0.043)	(0.016)	(0.012)	(0.044)
CNN-60k	0.644	0.740	0.906	0.146	0.268	0.792
RCN-1k	**0.891**	**0.947**	**0.950**	**0.747**	**0.719**	**0.919**
	(0.007)	(0.002)	(0.002)	(0.008)	(0.010)	(0.002)

The results clearly show that RCN outperforms CNN on all noisy test images, even though RCN only trained on 1k clean images and CNN trained on the full training dataset. The RCN classification accuracies are further improved when more training data is available. Fig. S22 summarizes the classification accuracies on level-3 noisy test images using RCN and CNN trained on different sets of data. For completeness, we also include RCN results when trained on the full dataset.

8.8 Occlusion reasoning on MNIST

8.8.1 Dataset

To test occlusion reasoning [51–53], we created a new variant of MNIST by superposing the image of a rectangular ring onto each digit image such that part of the rectangular ring occluded the digit and part of the digit occluded the rectangular ring. First, the MNIST images were upsampled by a factor of 4 to create images of size (112, 112). The rectangular ring had an overall width of 35 pixels and the width of the ring was 10 pixels. The position of the rectangle was chosen randomly in a (10, 10) window around the center of the image. After the placement of the ring, overlap regions between the ring and the digit were identified and the image modified such that some of the regions of the rectangle were behind the digit and vice versa. Some of the placements of the rectangle created a single overlap area with the digit and in those cases the digit was placed fully in front of the rectangle. Corresponding to each modified image, a mask image is created to record the ground truth occlusion

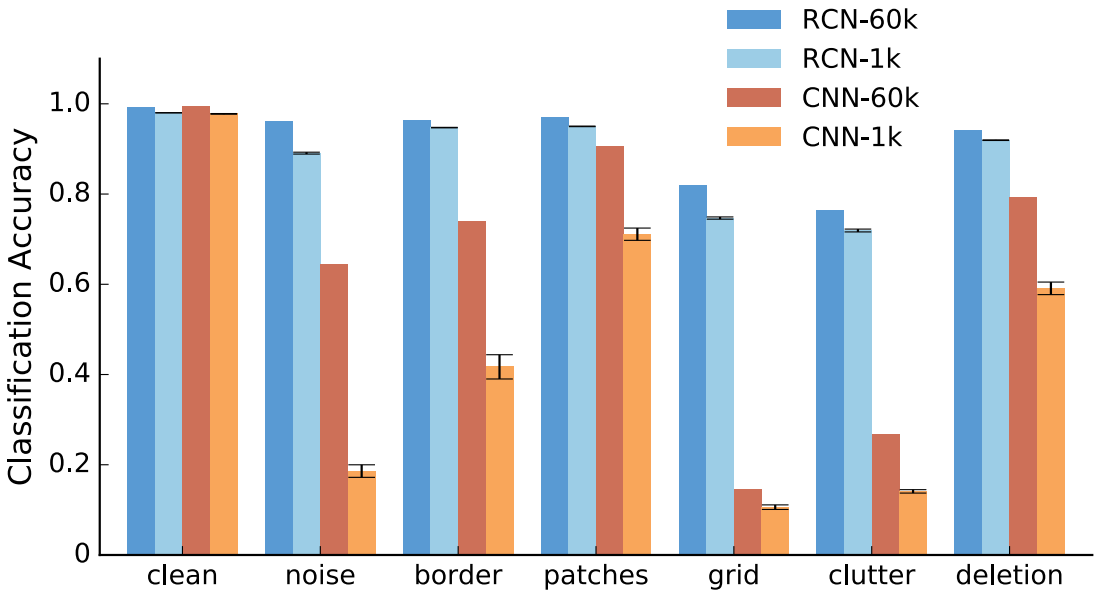


Figure S22: Classification results on noisy MNIST using RCN and CNN.

relations. An example of the occluded digit image is shown in Fig. S23(A), and its ground truth mask is shown in Fig. S23(G).

8.8.2 Classification and occlusion reasoning

The RCN network was trained on 11 categories: ten MNIST digits and the rectangular ring category. We used 20 examples per digit category and the single rectangle example for training. The parameter sets were identical to the previous experiment setting with 20 examples per category.

Classification accuracy without explicit occlusion reasoning is computed using the standard with-laterals forward and backward inference, and we report the best selected category that is not the rectangle.

Classification with explaining away (for occlusion reasoning) is achieved as follows. First, a with-laterals forward and backward inference is performed. We then select the top hypothesis from the result of this inference and explain away the contours from the input. The contour locations that are explained away are set to zero in the pre-processed edge outputs. Setting these to zero makes the likelihoods of presence and absence of contours to be equal, and it corresponds to explaining away interaction in an OR gate where one of the parents is set to ON. We then treat this modified edge output as a new input and do a second round of forward-backward inference in the network. If the best hypothesis from the first round of inference was a digit, then that digit is reported as the classification because explaining away the digit leaves only the rectangle as the unexplained object in the image. On the other hand, if the best hypothesis from the first inference is a rectangle, then the classification of the digit is set to the best category identified after explaining away the rectangle.

Identifying the occlusion regions and their occlusion relationships involves the masks and contours of the rectangle and digit hypotheses (that were identified in the two rounds of inference). First we identify the overlap regions between the predicted masks. For each overlapping region, the occlusion relation of the digit is identified using a simple heuristic that calculates whether the region owns its borders. For each region, we calculate the ratio of the length of the visible contours of the digit to the length of the predicted contours. A region that is occluded will have most of its contours invisible, whereas a region that is the occluder will have most of its contours visible. A visible-to-predicted length ratio of 0.1 was identified as effective using the validation set. We did not search exhaustively

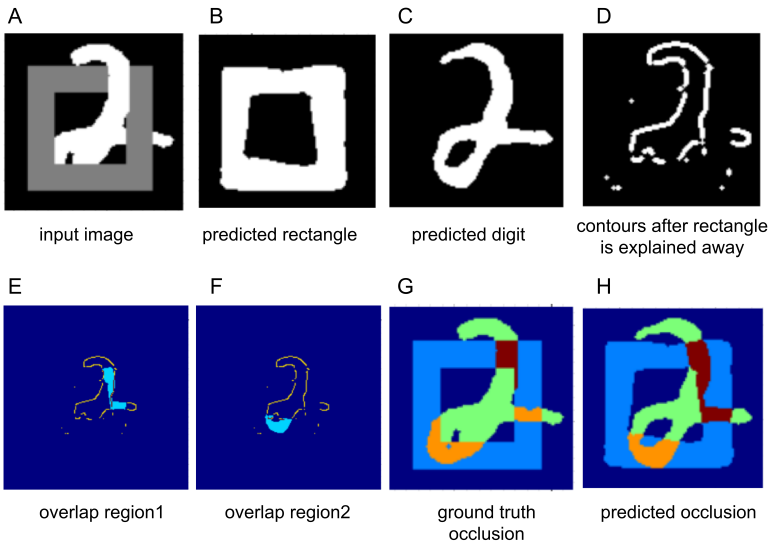


Figure S23: Different stages of occlusion reasoning with RCN.

over this parameter space to optimize the performance.

Fig. S23 shows details of the occlusion reasoning steps starting from the input image. Panels (E) and (F) show the two overlap regions, along with the visible contours of the digit. Region 1 is identified as an occluding rectangle and Region 2 is identified as being occluded by the rectangle. While Region 2's identification is correct, Region 1's identification is partially invalid because imperfections in the predicted mask created only two distinct regions of overlap compared to the three that are present in the ground truth (G).

Occlusion reasoning results are measured using the intersection over union (Jaccard index) of the identified areas in the test images with their counterparts in the ground truth image. All parameter adjustments were done on 1000 validation images, and the accuracy and occlusion reasoning results are reported based on 1000 test images. The model without explaining away yields 47.0% classification accuracy, and using explaining away improves the results substantially to 80.7%. The latter model successfully predicted the precise occlusion relations of the test images, obtaining a mean intersection over union (IOU) of 0.353 measured over the occluded regions.

8.9 Reconstruction from noisy MNIST using RCN, VAE, and DRAW

In this section we compare RCN with other generative models on the task of reconstructing noisy MNIST images (trained only on clean images). The first generative model is realized by a neural network and trained by means of the variational autoencoder (VAE) [49]. The architecture of the generative neural network is as follows: 20 latent variables with independent Gaussian priors are used as input, followed by two hidden layers of 500 neurons each with softplus activations, and finally a sigmoid to produce outputs in the $[0, 1]$ range. The recognition network takes an image as input and infers an approximate posterior over the 20 latent variables. Its structure is the same – two layers of 500 hidden units each with softplus activations. The output layer uses a linear activation and produces a pair of length 20 vectors, respectively describing the approximate posterior mean and variance of the latent variables. Training proceeds for 200 full epochs, using minibatches of size 100, and using the ADAM optimizer [108] with learning rate of 0.001. Without retraining, we reconstruct images from the clean and corrupted versions of the MNIST test set.

The second baseline generative model we consider is the DRAW network [50]. We used a reference implementation[22], with 25 recurrent steps, and trained with 10000 iterations. The batch size was 100,

[22] Available at `https://github.com/ericjang/draw`.

learning rate was 0.001, read/write glimpses were of size 5 by 5 pixels, and the number of hidden units was 256 (in both the encoder and the decoder).

The reconstruction results are reported in Tables S8, S10, and S12, for RCN, VAE, and DRAW, respectively. We show examples of the inputs and reconstructions for different types and levels of noise. Means and standard deviations of the reconstruction mean square errors (MSE) are reported respectively in Tables S9, S11, and S13 for quantitative evaluation. Reconstruction MSEs for the noise level 3 are summarized in Fig. S24 for a clearer comparison between different methods, including an additional RCN model trained on the full MNIST training dataset. As one can see, VAE and DRAW networks outperform RCN on the clean test data, but they do significantly worse on most noisy images. This is not surprising because, as can be seen in Table S12, DRAW is learning an overly flexible model that almost copies the input image in the reconstruction, which is desirable for clean images but very vulnerable to clutter and other types of noise. RCN's resilience against noise is also reflected by its much smaller standard errors of reconstruction MSE.

Table S8: RCN inputs and reconstructions on MNIST test data for different types and levels of noise.

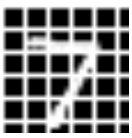
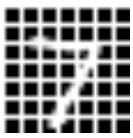
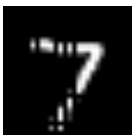
Noise type	Noise level						
	no noise	level 1		level 2		level 3	
	input	input	reconstr	input	reconstr	input	reconstr
clean							
noise							
border							
patches							
grid							
clutter							
deletion							

Table S9: Mean and standard deviation of reconstruction MSE for RCN. Computed over 10 independent runs, each of which included training the RCN and producing a reconstruction from the test set. Reconstruction error measured w.r.t. noiseless MNIST test images.

Noise type	Noise level			
	no noise	level 1	level 2	level 3
clean	$0.0163\ (\pm 3.8 \times 10^{-5})$			
noise		$0.0231\ (\pm 1.5 \times 10^{-4})$	$0.0279\ (\pm 2.8 \times 10^{-4})$	$0.0388\ (\pm 3.9 \times 10^{-4})$
border		$0.0232\ (\pm 3.7 \times 10^{-4})$	$0.0286\ (\pm 4.9 \times 10^{-4})$	$0.0365\ (\pm 4.8 \times 10^{-4})$
patches		$0.0244\ (\pm 8.6 \times 10^{-5})$	$0.0267\ (\pm 9.6 \times 10^{-5})$	$0.0289\ (\pm 1.2 \times 10^{-4})$
grid		$0.0835\ (\pm 2.2 \times 10^{-3})$	$0.0981\ (\pm 2.4 \times 10^{-3})$	$0.0850\ (\pm 1.7 \times 10^{-3})$
clutter		$0.0539\ (\pm 5.0 \times 10^{-4})$	$0.0628\ (\pm 6.3 \times 10^{-4})$	$0.0718\ (\pm 9.3 \times 10^{-4})$
deletion		$0.0264\ (\pm 7.2 \times 10^{-5})$	$0.0291\ (\pm 5.5 \times 10^{-5})$	$0.0318\ (\pm 7.2 \times 10^{-5})$

Table S10: VAE inputs and reconstructions on MNIST test data for different types and levels of noise.

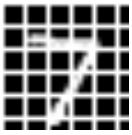
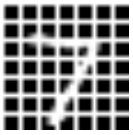
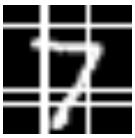
Noise type	Noise level						
	no noise	level 1		level 2		level 3	
	input	input	reconstr	input	reconstr	input	reconstr
clean							
noise							
border							
patches							
grid							
clutter							
deletion							

Table S11: Mean and standard deviation of reconstruction MSE for VAE. Computed over 10 independent runs, each of which included learning the VAE from the training set and producing a reconstruction from the test set. Reconstruction error measured w.r.t. noiseless MNIST test images.

Noise type	Noise level			
	no noise	level 1	level 2	level 3
clean	$0.0130\ (\pm 2.8 \times 10^{-04})$			
noise		$0.0544\ (\pm 2.4 \times 10^{-03})$	$0.0894\ (\pm 6.0 \times 10^{-03})$	$0.1380\ (\pm 1.2 \times 10^{-02})$
border		$0.1384\ (\pm 1.8 \times 10^{-02})$	$0.1470\ (\pm 1.6 \times 10^{-02})$	$0.1594\ (\pm 2.3 \times 10^{-02})$
patches		$0.0428\ (\pm 1.1 \times 10^{-03})$	$0.0518\ (\pm 1.4 \times 10^{-03})$	$0.0601\ (\pm 1.8 \times 10^{-03})$
grid		$0.1911\ (\pm 2.6 \times 10^{-02})$	$0.2065\ (\pm 2.3 \times 10^{-02})$	$0.1972\ (\pm 3.4 \times 10^{-02})$
clutter		$0.1290\ (\pm 8.6 \times 10^{-03})$	$0.1469\ (\pm 1.1 \times 10^{-02})$	$0.1613\ (\pm 1.2 \times 10^{-02})$
deletion		$0.0193\ (\pm 4.9 \times 10^{-04})$	$0.0224\ (\pm 5.9 \times 10^{-04})$	$0.0256\ (\pm 7.1 \times 10^{-04})$

Table S12: DRAW inputs and reconstructions on MNIST test data for different types and levels of noise.

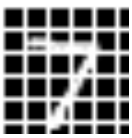
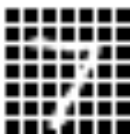
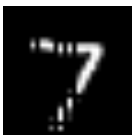
Noise type	Noise level						
	no noise	level 1		level 2		level 3	
	input	input	reconstr	input	reconstr	input	reconstr
clean							
noise							
border							
patches							
grid							
clutter							
deletion							

Table S13: Mean and standard deviation of reconstruction MSE for DRAW. Computed over 40 independent runs, each of which included training the DRAW and producing a reconstruction from the test set. Reconstruction error measured w.r.t. noiseless MNIST test images.

Noise type	Noise level			
	no noise	level 1	level 2	level 3
clean	$0.0024\ (\pm 4.0 \times 10^{-4})$			
bg noise		$0.0143\ (\pm 1.9 \times 10^{-3})$	$0.0327\ (\pm 4.6 \times 10^{-3})$	$0.0764\ (\pm 1.3 \times 10^{-2})$
boundary box		$0.1351\ (\pm 5.1 \times 10^{-2})$	$0.1737\ (\pm 6.4 \times 10^{-2})$	$0.2083\ (\pm 7.4 \times 10^{-2})$
box occlusion		$0.0220\ (\pm 1.3 \times 10^{-3})$	$0.0285\ (\pm 1.5 \times 10^{-3})$	$0.0347\ (\pm 1.8 \times 10^{-3})$
grid lines		$0.1494\ (\pm 3.9 \times 10^{-2})$	$0.1708\ (\pm 4.4 \times 10^{-2})$	$0.2154\ (\pm 6.1 \times 10^{-2})$
line clutter		$0.0920\ (\pm 9.7 \times 10^{-3})$	$0.1170\ (\pm 1.4 \times 10^{-2})$	$0.1401\ (\pm 1.9 \times 10^{-2})$
line deletion		$0.0175\ (\pm 5.5 \times 10^{-4})$	$0.0226\ (\pm 6.7 \times 10^{-4})$	$0.0274\ (\pm 7.8 \times 10^{-4})$

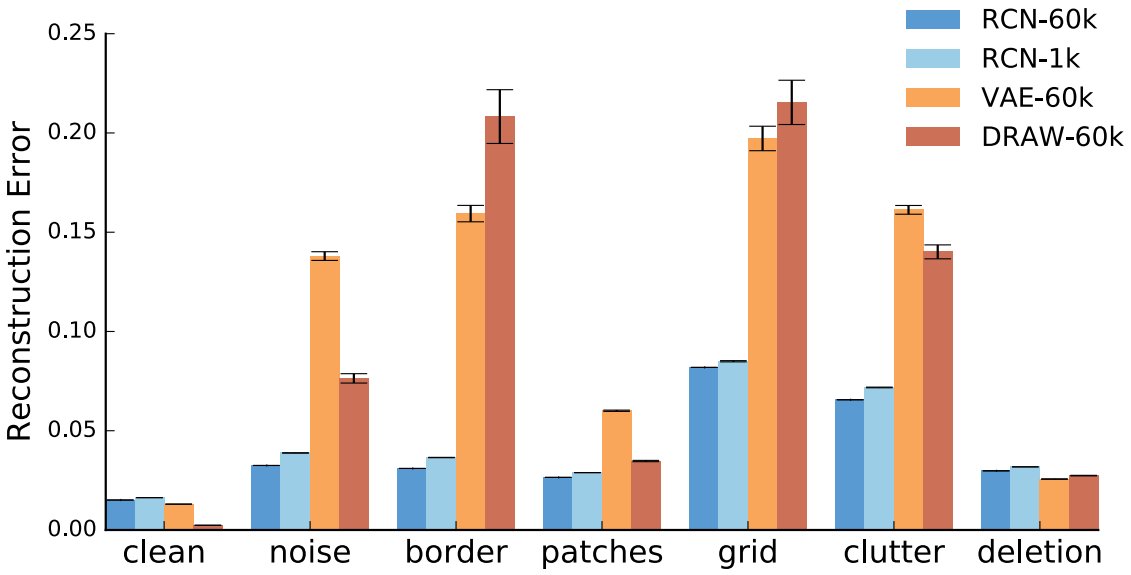


Figure S24: Reconstruction errors of noisy MNIST. Bars show the mean ± SEM.

8.10 Importance of lateral connections and backward pass

To test the importance of lateral connections in the RCN, we perform the following lesion experiments on the noiseless MNIST dataset. We followed the same set-up as in Section 8.7.2, and used the same parameters for all training scenarios. The previous results of RCN are obtained by a forward pass followed by a backward pass, both with lateral connections. To isolate the role of laterals and the backward pass, we perform the same experiments with three versions of RCN: (1) only forward pass, without laterals; (2) only forward pass, with laterals; (3) forward and backward passes, both without laterals. The classification performance on the full MNIST test dataset is shown in Table S14.

Table S14: Classification results on noiseless MNIST data.

# training examples per class	1	5	10	20	30	50	100
Pool size	57	45	45	37	37	33	25
Perturbation factor	1.0	1.0	2.0	2.0	2.0	2.0	2.0
FP (Lats OFF) (%)	13.81	24.85	26.31	40.19	42.97	54.36	79.85
FP (Lats ON) (%)	29.71	49.51	79.41	85.07	86.75	89.44	95.24
FP (Lats OFF) + BP (Lats OFF) (%)	38.92	66.16	64.86	80.82	80.92	86.75	95.30
FP (Lats ON) + BP (Lats ON) (%)	**77.23**	**92.53**	**95.49**	**96.70**	**97.29**	**97.56**	**97.98**

The results demonstrate that the roles of lateral connections and the backward pass are for achieving higher selectivity. Classification accuracies are lower with those mechanisms turned off, especially in the low sample complexity regime where the character models are more flexible.

8.11 The running time scaling of two-level and three-level RCN models

It is well-known that hierarchical compositional models with feature (or "part") sharing are much more efficient than single-layer models in terms of representation and inference (for example, [76]). This is also true for RCN models. The pooling allows for sharing of prototypical features on the lower layers by parent features on the higher layers, even in the existence of noise. This reusability significantly reduces the representational complexity of RCN. Interestingly, the deeper hierarchical

structures of RCN make its inference significantly more computationally efficient compared with the two-level hierarchy.

Consider a typical inference task of single-object classification in RCN, which includes a forward pass and a backward pass; more complicated inference tasks such as parsing can be analyzed similarly. The forward pass takes in the pre-processed image and outputs a set of hypotheses. Each hypothesis corresponds to a top-layer feature. In the backward pass, we clamp each highly ranked top-layer feature and perform layer-wise MAP inference to re-score the hypotheses to find the final winner.

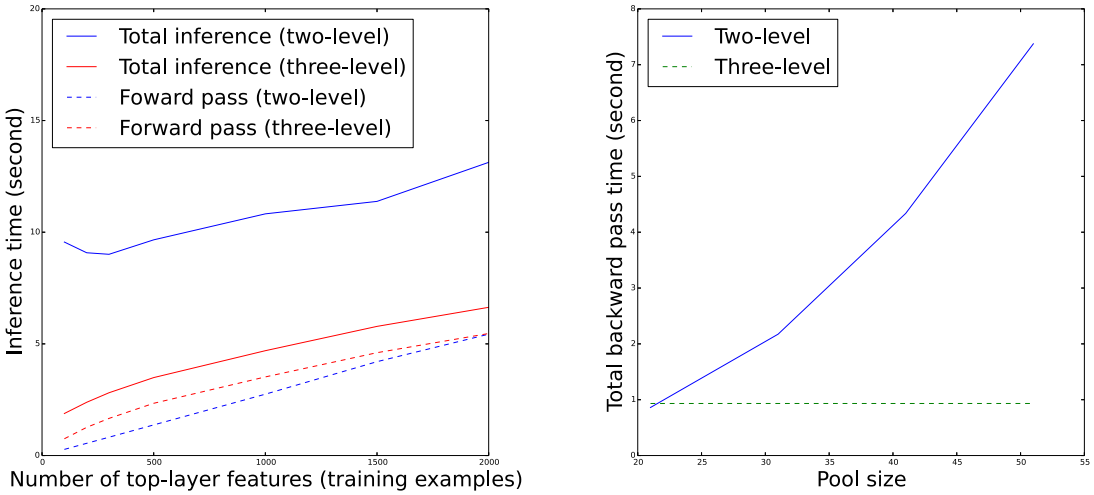
Next we consider the scaling behaviors of forward and backward pass complexities, in terms of the number of objects modeled by the RCN and the dimensionality of the state space that MAP inference was performed, respectively. In practice, both quantities must be large enough to capture the richness of variations that occur in real-world datasets.

The inference complexity of a forward pass in a given RCN layer depends linearly on both the number of parent features and the density of parent-child connections, similar to most neural networks. For modeling practical large-scale datasets, the top-layer of RCN, where each feature corresponds to an "object" (or a template of an object), has by far the greatest width. Hence in its asymptotic limit, we only have to consider the scaling behavior of the top-layer forward pass time. Due to the increased reusability of deeper hierarchical RCN (compared with shallower models), the representational complexity of the top-layer features in terms of the sub-object layer (the "part" layer) is much reduced. In other words, the connection between object and part layers is much sparser for hierarchical RCN models. Therefore, theoretically the forward pass time of RCNs with deeper hierarchies have a milder linear growth than shallower (two-level) models.

The backward pass involves solving an MAP inference problem, whose complexity is in general quadratic in the dimensionality of the state space. Depending on the parameterization of perturbation laterals, the dimensionality of the state space is at least linear in the number of pool states, and in the most general case quadratic – this is the case for square pools, where the number of pool states is quadratic in pool size. Therefore, when large pooling is required to capture the variations in the dataset, the backward pass inference in a two-level RCN model could be very expensive. However, for a model with more pooling layers, the total pooling is spread out into multiple layers, where the required pooling in each layer is reduced, making the layerwise backward pass inference more computationally efficient. Therefore, the overall backward pass inference complexity scales much better with respect to large pools in deeper hierarchies of RCN.

Next we perform experiments to empirically demonstrate the scaling phenomena mentioned above. We train two-level and three-level RCN models on the MNIST dataset (see Section 8.7 for details of the dataset), and vary the two key parameters – the total number of training examples from 100 to 2,000, and the pool size from 21 to 51. For the three-level hierarchy, we fix the lower-layer pool size to be 7, and choose the pool size of the top-layer such that the three-level hierarchy has the same total pooling as the two-level model. We test the RCN models on 100 test images of MNIST and record the times for the forward pass, backward pass, and total inference. Note that the backward pass time does not depend on the number of top-layer features, since we backtrace a fixed number of top hypotheses found by the forward pass – 20 in our experiments.

In Fig. S25(a), we plot the total inference times and forward pass times of the two-level and three-level models with respect to the total number of training examples. As expected by our previous analysis, the forward pass times of both models grow (piecewise-) linearly, and the curve of three-level hierarchy eventually becomes flatter than, and dominated by, the two-level model, after a burn-in period. In Fig. S25(b), we show the growth of backward pass time with respect to the pool size. Again, as expected, the backward pass time of two-level model increases much faster than that of the three-level hierarchy. In the large-pool regime, the computational advantage of the three-level hierarchy is more substantial.



(a) Scaling of inference time w.r.t. number of top-layer features (i.e. number of training examples).

(b) Scaling of backward pass time w.r.t. pool size.

Figure S25: Running time scaling for two-level and three-level hierarchical RCN models.

8.12 RCN on 3D object renderings

In this section, we will describe an experiment with RCN parsing scenes with 3D object renderings. This experiment serves to address the applicability of RCN to more general scene parsing tasks. The 3D object renderings correspond to everyday objects, labeled according to one of the following 10 categories: chair, table, airplane, skateboard, motorcycle, gun, headphones, shoe, toilet, and guitar. Training samples of this dataset are shown in Fig. S26.

Figure S26: Sample images of the objects training dataset. The categories are everyday objects: chair, table, airplane, skateboard, motorcycle, gun, headphones, shoe, toilet, and guitar.

In order to learn the features we use the procedure described in Section 5. During training, the model is presented with clean images of a single object against a gray background. The training data set contains, for each of the 10 categories, five different instances (different objects of that category), each of which is available in 12 viewpoints; this results in a total of 600 training examples. The test dataset contains complex scenes with object instances that were unseen in training. The scenes are cluttered arrangements of multiple objects (some partially occluded) against random scene backgrounds (Fig.

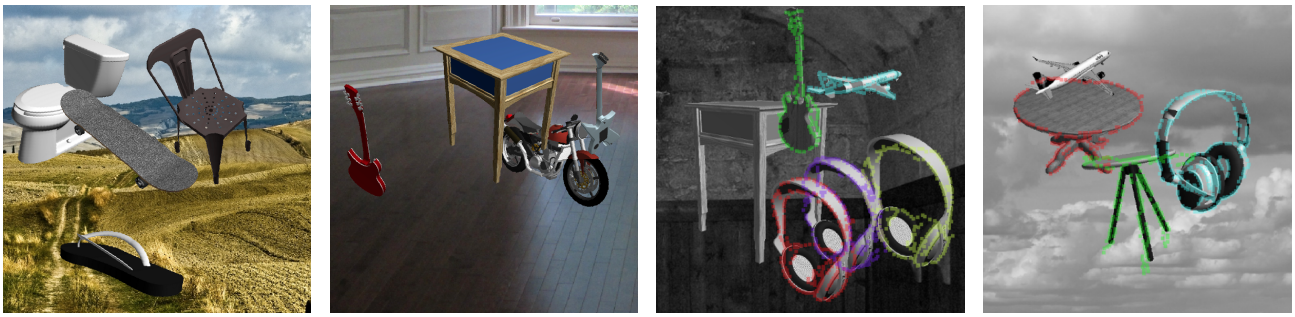


Figure S27: Left: Example images from the test set, where we use cluttered scenes of objects against random backgrounds. Note that in the test images there can be multiple categories represented and objects occlude one another. Right: Illustration of the 4-level model's backtrace inference pass. The colored outlines represent the model's object detections, where the colors identify specific objects. The left backtrace image shows five true positives, despite the difficult scene with identical, overlapping headphones instances. On the right we see the model correctly identified the tables and headphones objects, but missed the airplane.

S27). The ability of RCN to reliably classify objects without training on these full scenes with uncorrelated backgrounds and in the presence of occlusions exemplifies the strong data efficiency of the model.

Whereas in the case of scenes consisting of letters, we did not find any improvement in accuracy with hierarchies deeper than two levels, for 3D objects we found that a 4-level hierarchy gave the best performance. The models are evaluated for precision and recall at RCN's working point, and the results are shown in Table S15 for two tuned models. Because the inference result of RCN is multiple hypotheses with confidence levels, the key metric to consider is AP, the average value of precision as a function of recall. The 4-level hierarchy clearly outperforms the flat network. Fig. S27 (right) shows examples of the inferred contours (backtraces) resulting from the hierarchical model's backward pass.

Table S15: Detection results of RCNs on 3D object scenes.

Model	Precision	Recall	AP
2 level	0.328	0.448	0.413
4 level	0.426	0.680	0.621

It is important to highlight that RCN's backward pass is able to individually backtrace each of the detected instances – i.e., it is not performing semantic segmentation. In Fig. S27 (third image from the left), we can see how it is independently recovering the contours of three individual (and partially overlapping) headphones, corresponding to backtracing separate top-level hypotheses of the same training instance.

We make the following observations about RCN's performance on the 3D object data set:

- *False detections in clutter:* In backgrounds with large amounts of clutter, RCN produced false detections of objects. Structured clutter created more problems compared to random clutter.
- *Missed detections:* RCN missed objects when the foreground and background colors are too similar, missing on faint edge cues, and when the test instances had a different shape compared to the training instances.
- *Insufficient instance generalization:* RCN generalizes to the test instances using the flexibility of the contour hierarchy. For some objects (chairs, for example), this flexibiltiy was not sufficient

to generalize to a very different looking version of the object. Increasing the flexibility further has the downside of increasing the false detections.

- *Large number of training instances to accommodate the different viewpoints:* The model achieves some amount of 3D rotation invariance (approximately 20 degrees) through its translation pooling, but it still results in storing a large number of instances at the top level of the network.

For RCN to achieve state-of-the-art detection performance on general real-world images, it would need additional mechanisms that are not present in its current form. We discuss them in Section 8.13.

8.13 Improving RCN

Even though the experiment of Section 8.12 shows that the current incarnation of RCN can also be applied successfully to object detection and instance segmentation, RCN does not produce state-of-the-art results on real-world images such as those from ImageNet. As we have shown in previous sections, RCN derives distinct advantages from being a highly-structured generative model, but additional pieces are needed to make RCN's detection performance competitive in those benchmarks:

- Instance merging: RCN in default mode uses a template per training instance. We can reduce them to a subset (an active set of "support templates"), as done in the ICDAR experiment (Section 8.5). However, more advanced approaches would merge multiple instances corresponding to the same category, or merge instances dynamically. This is required to scale to a large number of categories.
- Use of appearance during the forward pass: Surface appearance is now only used after the backward pass. This means that appearance information (including textures) is not being used during the forward pass to improve detection (whereas CNNs do). Propagating appearance bottom-up is a requisite for high performance on appearance-rich images.
- Non-translational pooling: Pooling in RCN as of now is only translational. Pooling over other transformations (such as 3D rotation) of a basic patch would improve the generalization ability of RCN. This limitation is also shared by current CNNs.
- Foreground-background modeling: In real images, the background context provides invaluable information about the objects that are potentially present in a scene. This information is currently ignored by RCN, which models only the foreground objects and ignores the context. On the other extreme, CNNs aggregate foreground and background information without distinction (such that it is possible to fool a CNN by showing an object out of context). A better way to deal with this, consistent with the RCN philosophy, is to use a joint generative model for both foreground and background, allowing the background to probabilistically influence the perceived foreground.

References and Notes

1. B. M. Lake, R. Salakhutdinov, J. B. Tenenbaum, Human-level concept learning through probabilistic program induction. *Science* **350**, 1332–1338 (2015). doi:10.1126/science.aab3050 Medline

2. K. Chellapilla, P. Simard, "Using machine learning to break visual human interaction proofs (HIPs)," in *Advances in Neural Information Processing Systems 17)* (2004) pp. 265–272.

3. E. Bursztein, M. Martin, J. C. Mitchell, "Text-based CAPTCHA strengths and weaknesses," in *Proceedings of the 18th ACM Conference on Computer and Communications Security* (ACM, 2011), pp. 125–138.

4. G. Mori, J. Malik, "Recognizing objects in adversarial clutter: Breaking a visual CAPTCHA," in *2003 IEEE Conference on Computer Vision and Pattern Recognition* (IEEE Computer Society, 2003), pp. I-134–I-141.

5. V. Mansinghka, T. D. Kulkarni, Y. N. Perov, J. Tenenbaum, "Approximate bayesian image interpretation using generative probabilistic graphics programs," in *Advances in Neural Information Processing Systems 26* (2013), pages 1520–1528.

6. I. Goodfellow, Y. Bulatov, J. Ibarz, S. Arnoud, V. Shet, "Multi-digit number recognition from street view imagery using deep convolutional neural networks," paper presented at the International Conference on Learning Representations (ICLR) 2014, Banff, Canada, 14 to 16 April 2014.

7. E. Bursztein, J. Aigrain, A. Moscicki, J. C. Mitchell, "The End is nigh: Generic Solving of text-based CAPTCHAs," paper presented at the 8th USENIX Workshop on Offensive Technologies (WOOT '14), San Diego, CA, 19 August 2014.

8. D. R. Hofstadter, *Metamagical Themas: Questing for the Essence of Mind and Pattern* (Basic Books, 1985).

9. T. S. Lee, D. Mumford, Hierarchical Bayesian inference in the visual cortex. *JOSA A* **20**, 1434–1448 (2003). doi:10.1364/JOSAA.20.001434 Medline

10. T. L. Griffiths, N. Chater, C. Kemp, A. Perfors, J. B. Tenenbaum, Probabilistic models of cognition: Exploring representations and inductive biases. *Trends Cogn. Sci.* **14**, 357–364 (2010). doi:10.1016/j.tics.2010.05.004 Medline

11. T. S. Lee, "The visual system's internal model of the world," in *Proceedings of the IEEE*, vol. 103 (IEEE, 2015), pp. 1359–1378

12. D. Kersten, A. Yuille, Bayesian models of object perception. *Curr. Opin. Neurobiol.* **13**, 150–158 (2003). doi:10.1016/S0959-4388(03)00042-4 Medline

13. C. D. Gilbert, W. Li, Top-down influences on visual processing. *Nat. Rev. Neurosci.* **14**, 350–363 (2013). doi:10.1038/nrn3476 Medline

14. V. A. F. Lamme, P. R. Roelfsema, The distinct modes of vision offered by feedforward and recurrent processing. *Trends Neurosci.* **23**, 571–579 (2000). doi:10.1016/S0166-2236(00)01657-X Medline

15. P. R. Roelfsema, V. A. F. Lamme, H. Spekreijse, Object-based attention in the primary visual cortex of the macaque monkey. *Nature* **395**, 376–381 (1998). doi:10.1038/26475 Medline

16. E. H. Cohen, F. Tong, Neural mechanisms of object-based attention. *Cereb. Cortex* **25**, 1080–1092 (2015). doi:10.1093/cercor/bht303 Medline

17. D. J. Field, A. Hayes, R. F. Hess, Contour integration by the human visual system: Evidence for a local "association field". *Vision Res.* **33**, 173–193 (1993). doi:10.1016/0042-6989(93)90156-Q Medline

18. C. D. Gilbert, T. N. Wiesel, Columnar specificity of intrinsic horizontal and corticocortical connections in cat visual cortex. *J. Neurosci.* **9**, 2432–2442 (1989). Medline

19. E. Craft, H. Schütze, E. Niebur, R. von der Heydt, A neural model of figure-ground organization. *J. Neurophysiol.* **97**, 4310–4326 (2007). doi:10.1152/jn.00203.2007 Medline

20. V. A. F. Lamme, V. Rodriguez-Rodriguez, H. Spekreijse, Separate processing dynamics for texture elements, boundaries and surfaces in primary visual cortex of the macaque monkey. *Cereb. Cortex* **9**, 406–413 (1999). doi:10.1093/cercor/9.4.406 Medline

21. E. A. DeYoe, D. C. Van Essen, Concurrent processing streams in monkey visual cortex. *Trends Neurosci.* **11**, 219–226 (1988). doi:10.1016/0166-2236(88)90130-0 Medline

22. X. Huang, M. A. Paradiso, V1 response timing and surface filling-in. *J. Neurophysiol.* **100**, 539–547 (2008). doi:10.1152/jn.00997.2007 Medline

23. H. Zhou, H. S. Friedman, R. von der Heydt, Coding of border ownership in monkey visual cortex. *J. Neurosci.* **20**, 6594–6611 (2000). Medline

24. S.-C. Zhu, D. Mumford, *A Stochastic Grammar of Images* (Now Publishers, 2007).

25. L. L. Zhu, Y. Chen, A. Yuille, Recursive compositional models for vision: Description and review of recent work. *J. Math. Imaging Vis.* **41**, 122–146 (2011). doi:10.1007/s10851-011-0282-2

26. A. L. Yuille, "Towards a theory of compositional learning and encoding of objects," in *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)* (IEEE, 2011), pp. 1448–1455.

27. R. Salakhutdinov, J. B. Tenenbaum, A. Torralba, "Learning to learn with compound HD models," in *Advances in Neural Information Processing Systems 24* (2012), pp. 1–9.

28. Y. Chen, L. Zhu, C. Lin, A. Yuille, H. Zhang, "Rapid inference on a novel AND/OR graph for object detection, segmentation and parsing," in *Advances in Neural Information Processing Systems 20* (2007), pp. 289–296.

29. L. Zhu, A. Yuille, "Recursive compositional models: Representation, learning, and inference," In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (IEEE, 2009), p. 5.

30. Z. Tu, X. Chen, A. L. Yuille, S.-C. Zhu, Image parsing: Unifying segmentation, detection, and recognition. *Int. J. Comput. Vis.* **63**, 113–140 (2005). doi:10.1007/s11263-005-6642-x

31. S. Fidler, A. Leonardis, "Towards scalable representations of object categories: Learning a hierarchy of parts," in *2007 IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (IEEE, 2007).

32. Y. Jin, S. Geman, "Context and hierarchy in a probabilistic image model," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2 (IEEE, 2006), pp. 2145–2152.

33. Supplementary materials.

34. Z. Si, S.-C. Zhu, Learning AND-OR templates for object recognition and detection. *IEEE Trans. Pattern Anal. Mach. Intell.* **35**, 2189–2205 (2013).

35. S. Geman, Invariance and selectivity in the ventral visual pathway. *J. Physiol. Paris* **100**, 212–224 (2006). doi:10.1016/j.jphysparis.2007.01.001 Medline

36. T. Serre, L. Wolf, S. Bileschi, M. Riesenhuber, T. Poggio, Robust object recognition with cortex-like mechanisms. *IEEE Trans. Pattern Anal. Mach. Intell.* **29**, 411–426 (2007). doi:10.1109/TPAMI.2007.56 Medline

37. C. Guo, S.-C. Zhu, Y. N. Wu, Primal sketch: Integrating structure and texture. *Comput. Vis. Image Underst.* **106**, 5–19 (2007). doi:10.1016/j.cviu.2005.09.004

38. J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference* (Morgan Kaufmann, 1988).

39. T. Wu, S.-C. Zhu, A numerical study of the bottom-up and top-down inference processes in and-or graphs. *Int. J. Comput. Vis.* **93**, 226–252 (2011). doi:10.1007/s11263-010-0346-6

40. J. Xu, T. L. Wickramarathne, N. V. Chawla, Representing higher-order dependencies in networks. *Sci. Adv.* **2**, e1600028 (2016). doi:10.1126/sciadv.1600028 Medline

41. D. Sontag, A. Globerson, T. Jaakkola, "Introduction to dual decomposition for inference," in *Optimization for Machine Learning* (MIT Press, 2010), pp. 1–37.

42. E. Bienenstock, S. Geman, D. Potter, "Compositionality, MDL priors, and object recognition," in *Advances in Neural Information Processing Systems 10* (1997), pp. 838–844.

43. J. Tsotsos, A. Rothenstein, Computational models of visual attention. *Scholarpedia* **6**, 6201 (2011). doi:10.4249/scholarpedia.6201

44. B. A. Olshausen, C. H. Anderson, D. C. Van Essen, A neurobiological model of visual attention and invariant pattern recognition based on dynamic routing of information. *J. Neurosci.* **13**, 4700–4719 (1993). Medline

45. Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition. *Proc. IEEE* **86**, 2278–2324 (1998). doi:10.1109/5.726791

46. K. Simonyan, A. Zisserman, "Very deep convolutional networks for large-scale image recognition," paper presented at the International Conference on Learning Representations (ICLR) 2015, San Diego, CA, 7 to 9 May 2015.

47. J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, F.-F. Li, "Imagenet: A large-scale hierarchical image database," in *IEEE Conference on Computer Vision and Pattern Recognition* (IEEE, 2009), pp. 248–255.

48. A. Wong, A. L. Yuille, "One shot learning via compositions of meaningful patches," in *2015 IEEE International Conference on Computer Vision* (IEEE, 2015), pp. 1197–1205.

49. D. P. Kingma, M. Welling, "Stochastic gradient VB and the variational auto-encoder," paper presented at the International Conference on Learning Representations (ICLR) 2014, Banff, Canada, 14 to 16 April 2014.

50. K. Gregor, I. Danihelka, A. Graves, D. J. Rezende, D. Wierstra, "DRAW: A recurrent neural network for image generation," in *Proceedings of the 32nd International Conference on Machine Learning* (Proceedings of Machine Learning Research, 2015), pp. 1462–1471.

51. C. K. Williams, M. K. Titsias, Greedy learning of multiple objects in images using robust statistics and factorial learning. *Neural Comput.* **16**, 1039–1062 (2004). doi:10.1162/089976604773135096 Medline

52. T. Gao, B. Packer, D. Koller, "A segmentation-aware object detection model with occlusion handling," in *2011 IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (IEEE, 2011), pp. 1361–1368.

53. R. Fransens, C. Strecha, L. Van Gool, "A mean field EM-algorithm for coherent occlusion handling in MAP-estimation problems," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1 (IEEE, 2006) pp. 300–307.

54. D. Karatzas, F. Shafait, S. Uchida, M. Iwamura, L. Gomez i Bigorda, S. R. Mestre, J. Mas, D. F. Mota, J. A. Almazan, L.-P. de las Heras, "ICDAR 2013 Robust Reading Competition," in *2013 12th International Conference on Document Analysis and Recognition* (IEEE, 2013), pp. 1484–1493.

55. M. Jaderberg, K. Simonyan, A. Vedaldi, A. Zisserman, "Deep structured output learning for unconstrained text recognition," paper presented at the International Conference on Learning Representations (ICLR) 2015, San Diego, CA, 7 to 9 May 2015.

56. J. Wang, A. Yuille, "Semantic Part segmentation using compositional model combining shape and appearance," in *2015 IEEE Conference on Computer Vision and Pattern Recognition* (IEEE, 2015), pp. 1788–1797.

57. D. Tabernik, A. Leonardis, M. Boben, D. Skočaj, M. Kristan, Adding discriminative power to a generative hierarchical compositional model using histograms of compositions. *Comput. Vis. Image Underst.* **138**, 102–113 (2015). doi:10.1016/j.cviu.2015.04.006

58. S. M. Ali Eslami, N. Heess, T. Weber, Y. Tassa, K. Kavukcuoglu, "Attend, infer, repeat: Fast scene understanding with generative models," paper presented at the 30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain, 5 to 10 December 2016.

59. M. L'azaro-Gredilla, Y. Liu, D. S. Phoenix, D. George, "Hierarchical compositional feature learning," arXiv:1611.02252 [cs.LG] (7 November 2016).

60. S. Ullman, Visual routines. *Cognition* **18**, 97–159 (1984). doi:10.1016/0010-0277(84)90023-4 Medline

61. S. Litvak, S. Ullman, Cortical circuitry implementing graphical models. *Neural Comput.* **21**, 3010–3056 (2009). doi:10.1162/neco.2009.05-08-783 Medline

62. D. George, J. Hawkins, Towards a mathematical theory of cortical micro-circuits. *PLOS Comput. Biol.* **5**, e1000532 (2009). doi:10.1371/journal.pcbi.1000532 Medline

63. N. Le Roux, N. Heess, J. Shotton, J. Winn, Learning a generative model of images by factoring appearance and shape. *Neural Comput.* **23**, 593–650 (2011). doi:10.1162/NECO_a_00086 Medline

64. S. Eslami, C. Williams, "A generative model for parts-based object segmentation," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, K. Q. Weinberger, Eds. (Curran Associates, 2012), pp. 100–107.

65. A. Krizhevsky, I. Sutskever, G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25* (2012), pp. 1097–1105.

66. A. Globerson, T. S. Jaakkola, "Fixing max-product: Convergent message passing algorithms for MAP LP-relaxations," in *Advances in Neural Information Processing Systems 20* (2008), pp. 553–560.

67. V. Kolmogorov, Convergent tree-reweighted message passing for energy minimization. *IEEE Trans. Pattern Anal. Mach. Intell.* **28**, 1568–1583 (2006).

68. H. Wang, K. Daphne, Subproblem-tree calibration: A unified approach to max-product message passing. In *Proceedings of the 30th International Conference on Machine Learning* (Proceedings of Machine Learning Research, 2013), pp. 190–198).

69. S. E. Shimony, Finding MAPs for belief networks is NP-hard. *Artif. Intell.* **68**, 399–410 (1994). doi:10.1016/0004-3702(94)90072-8

70. T. Werner, A linear programming approach to max-sum problem: A review. *IEEE Trans. Pattern Anal. Mach. Intell.* **29**, 1165–1179 (2007). doi:10.1109/TPAMI.2007.1036 Medline

71. N. Komodakis, N. Paragios, G. Tziritas, MRF energy minimization and beyond via dual decomposition. *IEEE Trans. Pattern Anal. Mach. Intell.* **33**, 531–552 (2011).

72. T. Meltzer, A. Globerson, Y. Weiss, "Convergent message passing algorithms – a unifying view," In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, J. A. Bilmes, A. Y. Ng, Eds. (AUAI Press, 2009), pp. 393–401

73. S. G. Mallat, Zhifeng Zhang, Matching pursuits with time-frequency dictionaries. *IEEE Trans. Signal Process.* **41**, 3397–3415 (1993). doi:10.1109/78.258082

74. S. Fidler, M. Boben, A. Leonardis, "Optimization framework for learning a hierarchical shape vocabulary for object class detection," in *BMVC 2009* (2009).

75. B. Li *et al*., "Shrec'12 track: Generic 3d shape retrieval," paper presented at the Eurographics Workshop on 3D Object Retrieval, Cagliari, Italy, 13 May 2012.

76. A. Yuille, R. Mottaghi, Complexity of representation and inference in compositional models with part sharing. *J. Mach. Learn. Res.* **17**, 1–28 (2016).

77. M. A. Fischler, R. A. Elschlager, The representation and matching of pictorial structures. *IEEE Trans. Comput.* **22**, 67–92 (1973).

78. P. F. Felzenszwalb, D. P. Huttenlocher, Pictorial Structures for Object Recognition. *Int. J. Comput. Vis.* **61**, 55–79 (2005). doi:10.1023/B:VISI.0000042934.15159.49

79. L. L. Zhu, Y. Chen, A. Torralba, W. Freeman, A. Yuille, "Part and appearance sharing: Recursive compositional models for multi-view multi-object detection," in 2*010 IEEE Conference on Computer Vision and Pattern Recognition* (IEEE, 2010).

80. C. K. I. Williams, N. J. Adams, "DTs: Dynamic trees," in *Advances in Neural Information Processing Systems 12* (1999), pp. 634–640.

81. L. L. Zhu, C. Lin, H. Huang, Y. Chen, A. Yuille, "Unsupervised structure learning: Hierarchical recursive composition, suspicious coincidence and competitive exclusion," in *10th European Conference on Computer Vision* (Springer, 2008), pp. 759–773.

82. S. Ullman, Object recognition and segmentation by a fragment-based hierarchy. *Trends Cogn. Sci.* **11**, 58–64 (2007). doi:10.1016/j.tics.2006.11.009 Medline

83. L. Bottou, Y. Bengio, Y. Le Cun, "Global training of document processing systems using graph transformer networks," in *1997 IEEE Conference on Computer Vision and Pattern Recognition* (IEEE, 1997), pp. 489–494.

84. Z. Kourtzi, N. Kanwisher, Representation of perceived object shape by the human lateral occipital complex. *Science* **293**, 1506–1509 (2001). doi:10.1126/science.1061133 Medline

85. J. J. DiCarlo, D. Zoccolan, N. C. Rust, How does the brain solve visual object recognition? *Neuron* **73**, 415–434 (2012). doi:10.1016/j.neuron.2012.01.010 Medline

86. D. H. Hubel, T. N. Wiesel, Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *J. Physiol.* **160**, 106–154 (1962). doi:10.1113/jphysiol.1962.sp006837 Medline

87. L. Wiskott, T. J. Sejnowski, Slow feature analysis: Unsupervised learning of invariances. *Neural Comput.* **14**, 715–770 (2002). doi:10.1162/089976602317318938 Medline

88. R. D. S. Raizada, S. Grossberg, Towards a theory of the laminar architecture of cerebral cortex: Computational clues from the visual system. *Cereb. Cortex* **13**, 100–113 (2003). doi:10.1093/cercor/13.1.100 Medline

89. O. Ben-Shahar, S. Zucker, Geometrical computations explain projection patterns of long-range horizontal connections in visual cortex. *Neural Comput.* **16**, 445–476 (2004). doi:10.1162/089976604772744866 Medline

90. J. Hawkins, D. George, J. Niemasik, Sequence memory for prediction, inference and behaviour. *Philos. Trans. R. Soc. Lond. B Biol. Sci.* **364**, 1203–1209 (2009). doi:10.1098/rstb.2008.0322 Medline

91. K. Moutoussis, The physiology and psychophysics of the color-form relationship: A review. *Front. Psychol.* **6**, 1407 (2015). Medline

92. E. A. Lachica, P. D. Beck, V. A. Casagrande, Parallel pathways in macaque monkey striate cortex: Anatomically defined columns in layer III. *Proc. Natl. Acad. Sci. U.S.A.* **89**, 3566–3570 (1992). doi:10.1073/pnas.89.8.3566 Medline

93. F. Federer, J. M. Ichida, J. Jeffs, I. Schiessl, N. McLoughlin, A. Angelucci, Four projection streams from primate V1 to the cytochrome oxidase stripes of V2. *J. Neurosci.* **29**, 15455–15471 (2009). doi:10.1523/JNEUROSCI.1648-09.2009 Medline

94. C. W. Tyler, R. von der Heydt, "Contour-, surface-, and object-related coding in the visual cortex," in *Computer Vision: From Surfaces to 3D Objects* (Chapman and Hall/CRC, 2011), pp. 145–162.

95. F. T. Qiu, R. von der Heydt, Neural representation of transparent overlay. *Nat. Neurosci.* **10**, 283–284 (2007). doi:10.1038/nn1853 Medline

96. K. Friston, The free-energy principle: A unified brain theory? *Nat. Rev. Neurosci.* **11**, 127–138 (2010). doi:10.1038/nrn2787 Medline

97. D. J. Felleman, D. C. Van Essen, Distributed hierarchical processing in the primate cerebral cortex. *Cereb. Cortex* **1**, 1–47 (1991). doi:10.1093/cercor/1.1.1 Medline

98. A. M. Bastos, W. M. Usrey, R. A. Adams, G. R. Mangun, P. Fries, K. J. Friston, M. Andre, Canonical microcircuits for predictive coding. *Neuron* **76**, 695–711 (2013).

99. X. Lou, K. Kansky, W. Lehrach, C. C. Laan, B. Marthi, D. S. Phoenix, D. George, "Generative shape models: Joint text recognition and segmentation with very little training data," in *Advances in Neural Information Processing Systems 29* (2016).

100. L. von Ahn, B. Maurer, C. McMillen, D. Abraham, M. Blum, reCAPTCHA: Human-based character recognition via Web security measures. *Science* **321**, 1465–1468 (2008). doi:10.1126/science.1160379 Medline

101. K. He, X. Zhang, S. Ren, J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition* (IEEE, 2016), pp. 770–778.

102. M. Abadi *et al*., "TensorFlow: Large-scale machine learning on heterogeneous distributed systems." arXiv:1603.04467 [cs.DC] (14 March 2016).

103. I. Tsochantaridis, T. Hofmann, T. Joachims, Y. Altun, "Support vector machine learning for interdependent and structured output spaces," in *Proceedings of the Twenty-First International Conference on Machine Learnin*g (ACM, 2004), p. 104.

104. A. Bissacco, M. Cummins, Y. Netzer, H. Neven, "PhotoOCR: Reading text in uncontrolled conditions," in *2013 IEEE International Conference on Computer Vision*, (IEEE, 2013), pp. 785–792.

105. Z. Ghahramani, Probabilistic machine learning and artificial intelligence. *Nature* **521**, 452–459 (2015). doi:10.1038/nature14541 Medline

106. N. D. Goodman, J. B. Tenenbaum, T. Gerstenberg, "Concepts in a probabilistic language of thought," in *The Conceptual Mind: New Directions in the Study of Concepts*, E. Margolis, S. Lawrence, Eds. (MIT Press, 2015), pp. 623–654.

107. H. Lee, R. Grosse, R. Ranganath, A. Y. Ng, "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations," in *Proceedings of the 26th Annual International Conference on Machine Learning* (ACM, 2009), pp. 609–616.

108. D. Kingma, J. Ba, “Adam: A method for stochastic optimization,” paper presented at the International Conference on Learning Representations (ICLR) 2015, San Diego, CA, 7 to 9 May 2015.