Image segmentation via Cellular Automata

Mark Sandler[1], Andrey Zhmoginov[1], Liangcheng Luo[1], Alexander Mordvintsev[1], Ettore Randazzo[1], and Blaise Agúera y Arcas[1]

Google AI
{sandler,azhmogin,luolc,moralex,etr,blaisea}@google.com

Abstract. In this paper, we propose a new approach for building cellular automata to solve real-world segmentation problems. We design and train a cellular automaton that can successfully segment high-resolution images. We consider a colony that densely inhabits the pixel grid, and all cells are governed by a randomized update that uses the current state, the color, and the state of the 3×3 neighborhood. The space of possible rules is defined by a small neural network. The update rule is applied repeatedly in parallel to a large random subset of cells and after convergence is used to produce segmentation masks that are then back-propagated to learn the optimal update rules using standard gradient descent methods. We demonstrate that such models can be learned efficiently with only limited trajectory length and that they show remarkable ability to organize the information to produce a globally consistent segmentation result, using only local information exchange.
From a practical perspective, our approach allows us to build efficient models – our smallest automaton uses less than 10,000 parameters to solve complex segmentation tasks.

Keywords: Image segmentation, cellular automata, deep learning

1 Introduction

Cellular automata popularized by Conway's Game Of Life [7] and Stephen Wolfram's book "A New Kind of Science" [34] have been extensively studied and applied in various fields from physics and cryptography to dynamical systems and biology of multicellular organisms.

The core component of cellular automata (CA) is a local update rule defining a system transition over a single discrete time step. One of the most amazing properties, is that even the most straightforward time-independent update rules can result in very diverse and complex system behaviors. For example it has been known for a long time Conway's Game of Life is Turing complete [26]. In this work we turn away from hand-designed automata and instead learn the update rule using SGD. An alternative view of cellular automata is to think of cellular automata as a recurrent neural network. Such approach have been thoroughly explored as a model for sequence processing with applications ranging from natural language processing to reinforcement learning. Most widely used

recurrent cell types including RNNs, GRUs, LSTMs, and based on the iterative data processing with transformations that share their parameters. In recent years, recurrent architectures have also been explored in application to computer vision tasks ranging from object segmentation and salient region discovery to image super-resolution (see Section 2). The critical difference is that the vast majority of the prior work, use a small number of recurrent iterations and allow for long-range interactions via reduced resolution of deeper layers, thus departing from a traditional setting of a cellular automaton with local update rules. In this paper, we propose a novel architecture that uses a single update rule to iteratively arrive at the correct solution.

Our contributions are twofold. First, we demonstrate that a recurrent model with only *local* or quasi-local interactions (where we allow the usage of global statistics), can be trained to perform well in a highly nontrivial context-aware task of image segmentation. Secondly, we propose a set of techniques that allow us to limit the number of unroll steps and make it possible to stabilize the training of cellular automata over long periods. Perhaps what is even more interesting, with our techniques, trained models can adapt to a change of the underlying image, at least partially reusing information obtained while processing the original image. These models can thus be directly applicable to video processing.

The paper is organized as follows. In the next section we overview the related work. In section 3 we provide detailed design of our automata, including some of the technical decisions we had to make to stabilize the training. Finally in section 5 we present our experimental results.

2 Related work

Recurrent models in computer vision applications. Recurrent models so widely used in NLP have also been applied to computer vision tasks. Some work like [32] revolved around direct applications of sequence models to image processing. Another common theme is the idea that the same transformation can effectively be applied to different image scales, thus resulting in a recurrent CNN architecture. This approach has previously been applied to object classification [17,1], image super-resolution [13], label attribution [24,18] and salient region detection [33]. Similarly, in [2] authors explore recurrent convolutional layers as individual elements in a larger U-Net style network. In [36], recurrent CNN architecture emerges when approaching the label attribution problem using conditional random fields.

Hybrid models. Another natural application of recurrent convolutional models is in video processing or, more generally, in situations when the visual information has some form of temporal consistency. For example, in [25], recurrent CNNs are used for magnetic resonance imaging processing. In some approaches (see for example, [15,31,21,37,29,20,19]), single image representations produced by a CNN are fed into an RNN or LSTM-based recurrent model. A hybrid approach,

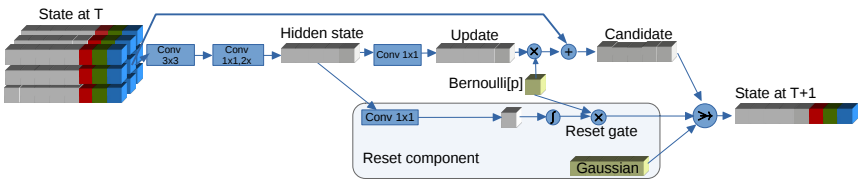


Fig. 1: A diagram of a single cell update rule. For non-resettable cells, the *reset component* is removed and the candidate always becomes the new cell state. The Bernoulli variable randomly disables changes to the cell to simulate asynchronous evolution of different cells. For resettable cells, the next state is a mixture (denoted by $\twoheadrightarrow$) of random noise and the candidate sate, with the reset gate determining the relative weight of each component.

where recurrent architectures are used for both the base CNN model and the processing of temporal information, can, for example, be found in [16].

Back to cellular automata. While all aforementioned applications inherit the core idea of recurrence from cellular automata, most depart from this simple model by introducing (explicitly or implicitly) non-local information exchange. Several recent papers including [9] turned back to the setting of the cellular automaton with the learned *local* recurrent update rules. Such systems have recently been explored in a range of topics from studying self-organization [22] to proposing a general computational platform [12,6].

The usage of cellular automata for image segmentation was also explored in earlier papers [3,5]. However the automaton considered in those works had a simple hand-designed parameterization used to detect similar regions, in a way similar to flood-fill type of algorithms. Here, in contrast we *learn* an automaton that can arrive to a very complex update rules.

3 Detailed Cellular Automaton Design

Following a common cellular automaton design similar for example to the one used in [22], we consider a grid space of size $h \times w$ inhabited by cells which can change their state based on the state of their closest neighbors and the state of the *environment* around them. In the following, we assume that each cell has a mutable state $s \in \mathbf{R}^d$. Then, for image processing tasks, we define the environment of a cell by associating it with color channels of the corresponding image pixel. More formally, a single step of the evolution of a cell state s corresponding to a pixel i is assumed to be given by:

$$s \leftarrow \mathcal{S}(\mathcal{N}(s), \mathcal{N}(i)),$$

where all cell states are assumed to be updated simultaneously, $\mathcal{N}$ denotes an immediate neighborhood of a cell or a pixel and $\mathcal{S} : \mathbf{R}^{nd} \times \mathbf{R}^{nk} \to \mathbf{R}^d$. Here k

is the size of the environment state ($k = 3$ for a standard RGB image) and n is the size of the neighborhood each cell is allowed to read. In this paper, we use the neighborhood of size 9 – that is 8 immediate neighbors plus the cell itself. The update rule $\mathcal{S}$ is a state transition that we *learn* with the goal of training the resulting cellular automaton to execute a specific task. We start by using update rules of the form $\mathcal{S}(s) = U(s) + s$, where $U(s)$ is a funciton that takes the neighborhood and produce state update. The space of functions we consider is 3-layer neural network consisting of single 3×3 convolution and 1×1. This update rule was inspired by residual blocks [10,35], however the critical difference is that rule is *the same* throughout the entire model. In experimental section we also explore variants of cell-updates without residual connections. In the latter case the update rule is simply $U(s)$.

We use C_t to denote the entire $h \times w \times d$ state at a given time t. The progression over t time steps is then equivalent to a repeated application of the transformation $\mathcal{S}$. That is

$$C_t = \mathcal{S}^t(C_0, \mathcal{I}) = \underbrace{[\mathcal{S} \circ \ldots \circ \mathcal{S}]}_{t \text{ times}}(C_0, \mathcal{I}),$$

where the same image $\mathcal{I}$ is provided at every step. For our experiments, we will use randomly initialized initial state C_0 sampled from standard Gaussian distribution.

We use a simple multi-layer neural network to compute the state update on each step. That is

$$\mathcal{S}(s, i) = [L_k \circ \cdots \circ L_1](\mathcal{N}(s), \mathcal{N}(i))$$

where each L_i is simply a fully connected layer with non-linearity that maps the state to a hidden representation. Since we use a dense colony, from a practical perspective, this can be efficiently implemented by applying the standard convolutional operator to the entire $h \times w \times (d + k)$ state. We note that in our architecture, only the first layer L_1 uses the information about the pixel neighbors. This highlights the fact that only minimal interaction between neighbors is required. The remaining operations are only using the current state. In our implementation, this translates to using 1×1 convolutions for all the layers except the first one.

In [22] it was suggested to further limit spatial interaction to a directional gradient. Instead, here we use learned spatial relationships for simplicity. Interestingly, the *actual* spatial relationship appears to be of only modest importance, and even using only random spatial depthwise convolution, the cellular automata can still perform the task fairly well. See section 5.3 for more details.

In its simplest variation, the output of the neural network is considered to be a proposed state update. The cell state is then incremented by this proposed update with a fixed probability p sampled independently for each pixel. This is done in a way similar to [22] to ensure the stability of the colony. In our experiments, we always use per-step update probability $p = 0.5$. In Section 3.3 we generalize this function to include the ability to control self-reset of the state.

3.1 Objective and training

Now that we defined the cellular automaton, how do we find the update rule $\mathcal{S}$? First of all, it is unrealistic to expect the colony to arrive at a good solution in one step. Instead, we pick a constant T, the *target* number of steps, after which we expect the colony to solve the problem. We can then use a standard cross-entropy loss function that is computed at every step of the colony once the colony has observed the image for longer than the initial threshold of $T_0 = T/3$. For each spatial location i with state s we can compute raw predictions $\hat{y} = \text{softmax}(W s_t)$ using the learned weights W and use the cross-entropy loss:

$$L(C_t, y) = -\mathbb{E}_i \sum_j y_j \log[\text{softmax}(W C_{i,t})]_j,$$

where the outer expectation is over all spatial locations and the inner summation is over all classes. The most natural training procedure widely used for training recurrent architectures is to unroll the cell state for T steps and use standard gradient descent with cell state s up to the number of steps and propagate the gradient through it. However, this approach results in a cellular colony overfitting to that particular number of steps T, and the prediction deteriorates before and after this step count. Even more crucially, the number of steps needed to produce good predictions in our experiments is fairly large (> 20). Training such unrolled cellular automaton requires large amounts of memory. In the remainder of this section, we describe additional techniques that allow us to significantly reduce the length of unrolls and improve overall model performance.

3.2 Mini-unrolls

Naive training of a cellular automaton with standard SGD looks very much like training a regular deep neural network. In particular, it requires (a) back-propagating through all T unrolled steps followed by (b) switching the images and resetting the cell state to process the following batch. As seen in Figure 11c, this approach to training a model leads to unstable outcomes where cellular automata diverge if allowed to proceed over a large number of steps. We therefore modify a single step of SGD by running it for only $K \ll T$ steps and then *reusing* a fraction of images and states after each mini-unroll. The probability p of reusing the state is chosen in such a way that only the fractions p_i and p_s of images and states correspondingly get reset before reaching the full-unroll target T. This achieves the following objectives:

1. Reset the image, but keep the state – allows the colony to learn to update image state preventing it from being frozen;
2. Keep the image and keep the state – allows the colony to stabilize the prediction for a larger number of steps;
3. Reset the state, either keep or reset the image – standard SGD-like step where we ensure that a colony can converge to a correct solution starting from random initialization.

To achieve the target reset probability p at step T, we use a per-unroll reset probability of $p^{K/T}$. This change allows us to scale the complexity and size of the colony without running out of memory, but also makes the colony capable of adapting to image changes. However, as shown in Figure 11c, the accuracy still deteriorates significantly in the long run. In the next section, we describe the approach that allows the colony to adapt without degradation continuously.

3.3 Reset gates for cell states

Cellular automaton design described in Section 3 converges when looked at the narrow task of predicting image after a fixed number of steps. In practice, we observed that the cell states tended to converge to quasi-stable equilibrium that slowly deteriorated over time. For instance, in Figure 11c the colony trained using standard SGD diverged after several hundred steps. Models trained this way also did not properly respond to a change of the underlying image. This latter property is important, for example, for video processing, but we observed that once the input image was updated, the cell state of the trained model slowly deteriorates. One such example is shown in Figure 11b where slight shifts to image lead to significant performance degradation. More formally, if C_0 is some initial cell state and $\mathcal{I}_0$ is the input image, then empirically we observe that once the colony converges to a stable state $S_{\mathcal{I}_0} = \mathcal{S}^*(C_0, \mathcal{I}_0)$

$$\mathcal{S}^\tau(s_{\mathcal{I}_0}, J) \sim s_{\mathcal{I}_0}$$

for an arbitrary image J and large values of $\tau \in \mathbb{N}$. Note that we use $\sim$ here to indicate that predictions for the states on both sides approximately match, while the states themselves may be different (and change over time).

This behavior is undesirable if we want the colony to properly evolve in response to a changing input. To address this, we introduce a "reset gate" reminiscent to that of GRU [8] recurrent units, where a cell state can be reset based on its state. However one difference is that we use continuous source of randomness rather than resetting the state to zero. The new resettable state update $\mathcal{S}_r$ function now looks as follows.

$$r(s) = \sigma[W_R H(s)]$$
$$\mathcal{S}_r(s) = r(s)z + (U(s) + s)(1 - r(s))$$

where $z \in R^d$ is a random normal variable that provides independent random noise for each cell, $H(S)$ is a hidden state -the last layer before update, and W_R is a trainable $d \times 1$ matrix describing the transformation from the hidden state to a scalar indicating whether the cell should reset. The diagram of the full resettable cell is shown in Figure 1.

3.4 Cellular automata for high-resolution images

Applying our method as described to high-resolution images would require excessive computational resources and lead to a very large memory footprint during

training. While it is a departure from our previous design, we found that the most accurate and scalable approach to dealing with high-resolution images is to use CA state resolution that is smaller than the image resolution. In particular, we used convolutions and transpose convolutions with a finite stride to: (a) reduce the input image size to make it compatible with the CA state resolution and (b) transform the CA state into a high-resolution segmentation map. Experimental results with this approach are presented in Section 5.4.

3.5 State Normalization

Our CA models converge well even when no cell state normalization is used. However, we generally observed that normalization further stabilized optimization. We experimented with four types of normalization: batch-normalization [11] in the training mode and instance normalization [30], channel normalization [4] and no normalization at all. In our experiments, we found that the normalization methods often perform comparably, however have different limitations in practice. For instance for batch normalization we had to rely on the live mini-batch statistics even in evaluation mode to avoid state-shift caused by the cell state evolution.

4 Neural networks vs. Cellular Automata

As described in 3, the proposed cellular automata utilize the same type of computation as regular neural networks. However, the critical difference is how the information is propagated. In standard neural network architectures, a common intuitive picture implies that the deeper layers specialize in higher-level concepts using a different set of learned variables. In the case of a cellular automaton, the deeper layers are actually the same as earlier layers – and they all just evolve the state on the grid.

An alternative view is to treat these cellular models as recurrent convolutional models [28,2,24], where the output of each sequence of loops is fed into the model. However, such intuition does not capture the local-organization aspect. Indeed, the state cell on each step is only ever affected by the immediate neighbors. Thus, it is imperative for the model to propagate local information across many steps to achieve spatial consistency.

A variant of such local propagation is also exploited in *Isometric Networks* [27], where all layers have fixed spatial dimensions, and no layers with stride are used. This, similarly to present work forced the information propagation locality. However, in [27], there was no recurrence that allowed iterative incrementality of computation that cellular automata allow.

5 Experiments

In this section we present qualitative and quantitative results of applying our cellular automata to the Oxford Cats and Dogs Dataset [23]. This dataset contains about 3400 training and 3400 evaluation samples. The goal of our cellular

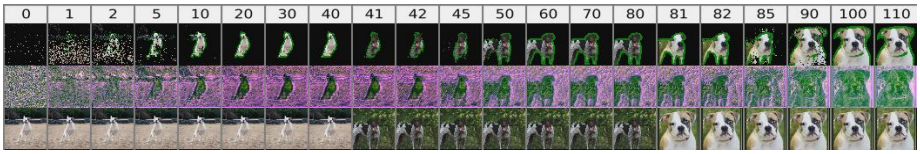


Fig. 2: Evolution of the colony with resettable cells where the underlying images change every 40 steps. The first row shows the prediction. The second row is a projection of the hidden state on RGB. Note how the predicted shape adopts to the image.

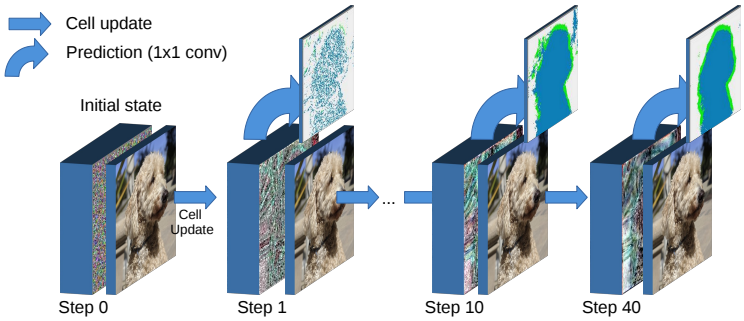


Fig. 3: Evolution of a cell colony for segmentation. Each step apples the same update rule (see Figure 1) to each cell independently.

automata was to segment the image pixels into "background", "object" and "object boundary" regions.

First, we describe our architectural setup. Unless noted otherwise, the length of mini-unroll of 10 steps and the target prediction goal of 40 steps in all our experiments. Each cell is described by a 48-dimensional state and contains 4 hidden layers. Inside the sequence of hidden layers, we use a state of size 64, i.e., the first convolutional operator uses transforms from 48 to 64 channels. The first layer in these sequence is a 3×3 convolution and the remaining layers are all 1×1 convolutions. The prediction is obtained by applying an additional 1×1 convolutional layer to the automaton state to get *num-classes* prediction vector for each pixel. We use RELU for all hidden layers, and use linear activations as is standard for logit predictions. We use instance normalization [30] for most of our experiments and standard Adam optimizer [14] with the learning rate $3 \cdot 10^{-4}$ and a batch size of 32. We train all our runs for 1 million steps.

5.1 Qualitative Experiments

We show prediction results of a trained colony on a validation sample of 16 images in Figure 3. The original images are shown in Figure 8a. As we can see in Figure 3, the prediction effectively starts close to random at step 1. By step

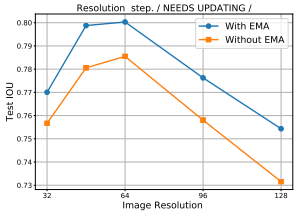


Fig. 4: Image size vs. IOU

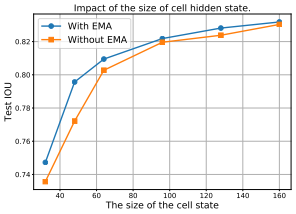


Fig. 5: Cell state vs. IOU

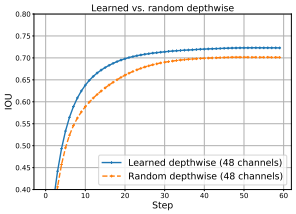


Fig. 6: Random vs. learned

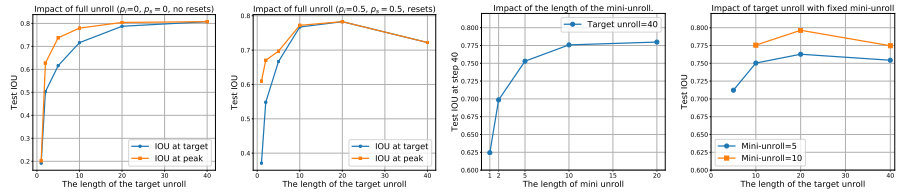
10 model predictions take the correct shape and the model uses the remaining steps to refine the prediction.

5.2 Ability to adapt to image changes (shifts and resets)

In Figure 11 we show how different models adapt to different types of image changes. The full-unroll model is essentially a model that is trained like a regular neural network with random sample at each batch and a full reset of the state. As one can see, the model rapidly reaches its highest accuracy, but then rapidly degrades to a random prediction. On the other hand, the non-resettable model that maintains some components of the state with images that change as described in Section 3.2 shows a much more gradual degradation. However, after a few image changes it too degrades severely. Finally, after soft reset gates are introduced, the model becomes fully stable.

5.3 Ablation study

Impact of the hidden state and the colony resolution An obvious question is how important is the size of the cell state. Note that in our experiments we have the state size and the size of the hidden layer linked together. So increasing the state size also automatically increases the size of hidden state. It can be seen in Figure 5 that the accuracy decreases fairly quickly when the size of the state

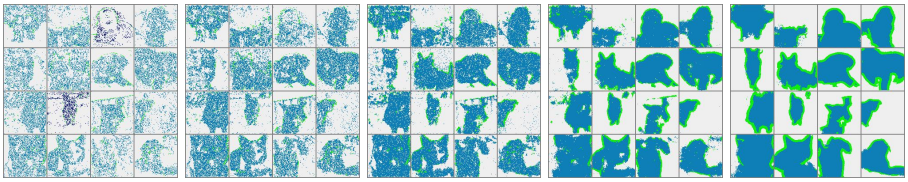


(a) Classic (Full) (b) Mini=Target (c) Target unroll 40 (d) Mini unroll 5/10

Fig. 7: Impact of target and mini-unrolls. Figure 7a corresponds to a standard SGD training with no mini-unroll and no cross-step information propagation. For 7c and 7d, the accuracy is always measured at the 40-step boundary.

(a) Original (b) d=48 (c) d=64 (d) d=96 (e) d=128

Fig. 8: Samples of predictions of colonies trained to segment at different resolutions d. All predictions are after 40 steps.



(a) 1 step (b) 2 steps (c) 4 steps (d) 10 steps (e) 40 steps

Fig. 9: Intermediate predictions after different number of steps.

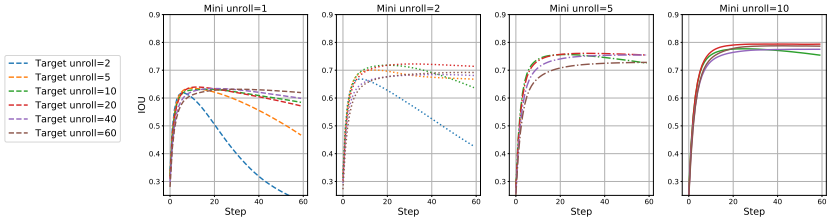


Fig. 10: Evolution of IOU accuracy per colony step

(a) Images change every 40 steps. (b) Images are randomly shifted every 10 steps. (c) Evolution of colony over stable image.

Fig. 11: IOU performance when underlying image changes for different types of models. The resettable and non-resettable models preserve the state and the images with probability 0.5 and use short unroll of size 10. Full unroll model uses classical SGD with full state and image resets. Note how introducing reset gate essentially stabilized the performance over a large number of evolution steps.

drops below 32, and generally increases with the increased size. In Figure 4, we show the impact of the image resolution on the IOU. Here, the increased image resolution makes it harder for the colony to produce correct results. We show some samples of actual classifications in Figure 8. To address this issue for higher resolution we show some preliminary results in Section 5.4.

Unroll length A natural question is how important is the number of steps over which the colony evolves. We have two parameters controlling this:

- *Mini unrolls* is the length of unroll that is done in a single step and back-propagated through and
- *Target unroll* is the target unroll length that is used to measure loss.

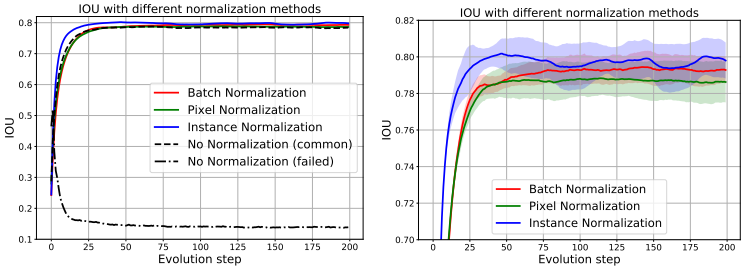
One important consideration is that when we use mini-unroll that mismatches the target unroll, we always have to modify the training procedure to preserve most of the images and cell state between training steps, because otherwise the training procedure will never observe cell states "old enough". These two hyper-parameters p_i and p_s, are defined in section 3.2. For experiments with the unroll length we used $p_i = p_s = 0.5$, however we note that results are not very sensitive to these parameters as long as both p_i and p_s are bounded away from 0 and 1. The impact of different unroll lengths and target unrolls is shown in Figure 11c. As can be seen from it, very short mini-unroll lengths (< 3) lead to both very low accuracy and quality degradation over time, while the longer mini-unrolls and higher target unrolls lead to stable outcomes. All the models in this study were resettable models. Figure 10 shows that increasing mini-unroll essentially saturates after 10-15 steps.

On the other hand, 7b shows the performance over short term is best when we use monolithic training. While that approach results in a slightly higher accuracy, the resulting colony is susceptible to slow degradation over longer periods as shown in Figure 11. Also the importance of resettable state can be clearly seen in Figure 3, where the resettable colony does not show any degradation over long term.

Different normalization As mentioned in section 3.5, our models can converge well without using any normalization, while introducing it can help stabilize optimization and improve generalization ability. In this section we conduct experiments on using different normalization methods and illustrate the results in Fig. 12.

We train the CA models with batch, pixel, instance and no normalization respectively, and evaluate them on the test data for multiple times. We can see in Fig. 12a that commonly the model trained without normalization already has good performance in terms of IOU. However, it may sometime fails disastrously, which indicates its instability and poor generalization.

This problem can be solved by state normalization. Fig. 12b shows the IOU curve using different normalization methods with error regions. The shaded region of each model indicates the standard deviation across its three independent



(a) IOU (no error regions). (b) IOU (with error regions).

Fig. 12: The impact of normalization methods in cellular automata. Both figures plot the average IOU curve of different normalization methods on three independent runs. Fig. 12a includes the results on two independent runs without normalization, where one of them failed. Fig. 12b shows the error regions.

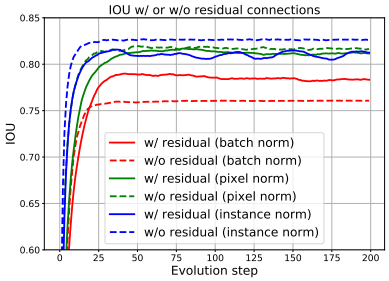


Fig. 13: The impact of residual connections.

Fig. 14: (In progress.) 96×96 model with a 48×48 internal state (see Section 5.4) vs. original 48×48 and 96×96. Note: (a) is $\approx 4\times$ faster than (c).

runs. These models with normalization are much more stable than the one without, and their performance are relatively similar to each other regarding the major overlap among the error regions.

Importance of residual connections Recall that we use a residual connection in the update rule of our cellular automation, i.e. $\mathcal{S}(s) = U(s)+s$, for its widespread success among many applications in segmentation tasks. But since our model is quite different from the conventional ones that we use the same rule for all the updating phases, the residual connection might behave differently from that in common models as well. Therefore, in this section we conduct experiments to show the impact of residual connections.

To construct the model without residual connection, we use an update rule of $\mathcal{S}(s) = U(s)$ where $U(s)$ has the same definition with $U(s)$ in the residual-style rule. We run the CA model and the no-residual variant on a batch of test data and show the results in Fig. 13. Interestingly, unlike in other con-

ventional models, the CA model without residual connection also has relatively good performance, and introducing the residual does not consistently guarantee an improvement in terms of IOU. Specifically, residual connection can enhance the model with batch normalization, while the no-residual variant has similar or even slightly better performance on the model using pixel or instance normalization. Generally speaking, our model can achieve acceptable performance in both different settings. The reason why it can behave well without a residual-style design remains an open question and we leave it as future work.

Importance of learned edge detectors In this section we discuss an experiment that measured the importance of the learned *spatial* filters. It has been shown [22] that even basic spatial filters work very well. We take it one step further and show that even *random* filters work reasonably well, suggesting that the internal transformation of the colony is the critical component that can adopt to arbitrary spatial combination rules. The results are shown in Figure 6.

5.4 High resolution experiments

The approach outlined in Section 3.4 allowed us to scale our CA models to the 384×384 input image size and leverage information contained in a higher-resolution image. In one experiment on the Oxford Cats and Dogs dataset, we compared the original CA acting on a 48×48 state given a 48×48 input image with another CA acting on a state of the same size, but provided with a 96×96 image. The 96×96 image was transformed with a kernel-size-3 stride-2 convolution into a $48 \times 48 \times 8$ input tensor that was then used by the CA. Similarly, each CA state was mapped into a 96×96 prediction map via a kernel-size-3 stride-2 transpose convolution. While adding only about 3500 parameters, 96×96 model improves IOU by $\approx 3\%$, see Figure 14. Interestingly, this 96×96 model also outperforms the original 96×96 CA model while being almost a factor of 4 faster.

5.5 Regime change in resettable models

One interesting property that we discovered in resettable cellular automata is that they almost always undergo a regime change at some seemingly random point during training. In Figure 15, we show how the average value of the reset gate (averaged over each pixel) changes during training. Notice that transition to a new regime is accompanied by a dramatic change in the model long-term stability. Understanding this phenomenon will be the subject of future work.

5.6 Model size

One remarkable property of a cellular automaton is its size. For instance, our model with a hidden channel size of 48 and 3 layers in each update step uses only about 50K parameters. A modification of this cellular automaton with a

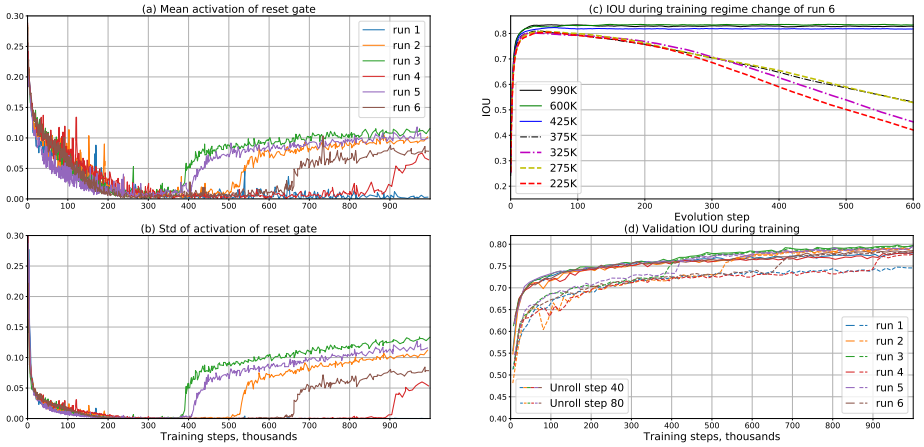


Fig. 15: Regime change of resettable gates. Curves in (a) and (b) correspond to gating activations in 6 identical runs (1M training steps). Figure (c) shows IOU of the run (6) at different training steps: before (dashed lines) and after the reset regime change (solid lines). Figure (d) shows the accuracy transition of all runs during training. Note how the accuracy of unroll step 79, exhibits a dramatic increase once the reset gate underwent the regime change during training.

first full convolutional layer replaced by a combination of a 1×1 convolution and a $3x3$ depthwise convolutions has a similar performance, but uses only 18K parameters. Reducing the number of channels to 32 leads to an even smaller network with ∼9K parameters and only moderate degradation in performance.

In table 1 we present the comparative trade off accuracy numbers depending on the size of the model and whether we use depthwise or 3x3 convolutions. In this table we include both the object IOU as well as the IOU of the boundary. The depthwise version of our cellular automata has nearly 3 times fewer parameters while having comparable performance at many models sizes.

6 Conclusions and Future Work

In this work we demonstrated that cellular automata can be trained to solve complex image segmentation tasks. While these automata can also be seen as neural networks, their distinctive property is the fact that each colony evolves by repeatedly applying *the same update rule*. In order to successfully train such models, we have introduced several new techniques that stabilized the training and enabled us to use short unrolls. From a practical perspective, these models have multiple advantages over classical neural networks. First, they provide a very natural framework for incremental computation where underlying image can change without having to discard intermediate computations. Thus, it naturally enables us to apply segmentation on continuously changing data (such as video). Second, the asynchronous spatial updates they can be implemented by a grid

Table 1: Model size vs. accuracy trade off. Comparison between using 1x1 + planar (depthwise) convolution vs standard 3x3 convolution as a first layer (other layers are always 1x1 convolutions and using different using different normalization methods. We use $(*)$ to indicate unstable runs.

Norm	Cell size	Hidden size	Depthwise			3×3 convolution		
			Params	Boundary	Object	Params	Boundary	Object
None	32	48	8.4K	42.7	76	21.4K	44.5	75
Batch				42	75		43.7	73
Inst				39	71		39	74
None	48	72	18.3K	46	79	47K	46	77.2
Batch				42.5	78		46.4	76
Inst				*	*		45	78
None	64	96	32.3K	48	80	82K	47.4	77.5
Batch				46	79		48.5	78
Inst				*	*		48.6	80.7

of processors without global synchronization or global data routing. Third, the resulting models are remarkably compact – less than 10K parameters for some variations.

From the theoretical perspective, we believe that cellular automata provide a promising framework, in which we can reason about how we can solve complex tasks by using multiple simple independent agents. For example our CA design can easily generalize to allow for richer interactions. For instance cells can be allowed to copy themselves to move around the grid, or leave breadcrumbs for other cells to utilize. Training such CA remains subject of further work.

References

1. Alom, M.Z., Hasan, M., Yakopcic, C., Taha, T.M.: Inception recurrent convolutional neural network for object recognition. CoRR **abs/1704.07709** (2017), http://arxiv.org/abs/1704.07709
2. Alom, M.Z., Hasan, M., Yakopcic, C., Taha, T.M., Asari, V.K.: Recurrent residual convolutional neural network based on u-net (r2u-net) for medical image segmentation. CoRR **abs/1802.06955** (2018), http://arxiv.org/abs/1802.06955
3. Andreica, A., Diosan, L., Voiculescu, I.: Parameterized cellular automata in image segmentation. In: 2016 18th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC). pp. 199–205 (2016)
4. Ba, J.L., Kiros, J.R., Hinton, G.E.: Layer normalization (2016)
5. Dioşan, L., Andreica, A., Boros, I., Voiculescu, I.: Avenues for the use of cellular automata in image segmentation. In: Squillero, G., Sim, K. (eds.) Applications of Evolutionary Computation. pp. 282–296. Springer International Publishing, Cham (2017)
6. Freivalds, K., Liepins, R.: Improving the neural GPU architecture for algorithm learning. CoRR **abs/1702.08727** (2017), http://arxiv.org/abs/1702.08727
7. Garner, M.: Mathematical games. the fantastic combinations of john conways new solitaire game life. Scientific American **223**, 120–123 (1970)
8. Gers, F., Schmidhuber, J., Cummins, F.: Learning to forget: Continual prediction with lstm. Neural computation **12**, 2451–71 (10 2000). https://doi.org/10.1162/089976600300015015
9. Gilpin, W.: Cellular automata as convolutional neural networks. CoRR **abs/1809.02942** (2018), http://arxiv.org/abs/1809.02942
10. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. CoRR **abs/1512.03385** (2015), http://arxiv.org/abs/1512.03385
11. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. CoRR **abs/1502.03167** (2015), http://arxiv.org/abs/1502.03167
12. Kaiser, L., Sutskever, I.: Neural gpus learn algorithms. In: Bengio, Y., LeCun, Y. (eds.) 4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings (2016), http://arxiv.org/abs/1511.08228
13. Kim, J., Lee, J.K., Lee, K.M.: Deeply-recursive convolutional network for image super-resolution. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016. pp. 1637–1645. IEEE Computer Society (2016). https://doi.org/10.1109/CVPR.2016.181, https://doi.org/10.1109/CVPR.2016.181
14. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. CoRR **abs/1412.6980** (2014)
15. Kuen, J., Wang, Z., Wang, G.: Recurrent attentional networks for saliency detection. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016. pp. 3668–3677. IEEE Computer Society (2016). https://doi.org/10.1109/CVPR.2016.399, https://doi.org/10.1109/CVPR.2016.399
16. Lee, C., Osindero, S.: Recursive recurrent nets with attention modeling for OCR in the wild. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016. pp. 2231–2239. IEEE Computer Society (2016). https://doi.org/10.1109/CVPR.2016.245, https://doi.org/10.1109/CVPR.2016.245

17. Liang, M., Hu, X.: Recurrent convolutional neural network for object recognition. In: IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015. pp. 3367–3375. IEEE Computer Society (2015). https://doi.org/10.1109/CVPR.2015.7298958, https://doi.org/10.1109/CVPR.2015.7298958
18. Liang, M., Hu, X., Zhang, B.: Convolutional neural networks with intra-layer recurrent connections for scene labeling. In: Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada. pp. 937–945 (2015), http://papers.nips.cc/paper/5634-convolutional-neural-networks-with-intra-layer-recurrent-connections-for-scene-labeling
19. Liu, M., Zhu, M.: Mobile video object detection with temporally-aware feature maps. In: 2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018. pp. 5686–5695. IEEE Computer Society (2018). https://doi.org/10.1109/CVPR.2018.00596, http://openaccess.thecvf.com/content_cvpr_2018/html/Liu_Mobile_Video_Object_CVPR_2018_paper.html
20. Liu, M., Zhu, M., White, M., Li, Y., Kalenichenko, D.: Looking fast and slow: Memory-guided mobile video object detection. CoRR **abs/1903.10172** (2019), http://arxiv.org/abs/1903.10172
21. McLaughlin, N., del Rincón, J.M., Miller, P.C.: Recurrent convolutional network for video-based person re-identification. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016. pp. 1325–1334. IEEE Computer Society (2016). https://doi.org/10.1109/CVPR.2016.148, https://doi.org/10.1109/CVPR.2016.148
22. Mordvintsev, A., Randazzo, E., Niklasson, E., Levin, M.: Growing neural cellular automata. Distill (2020). https://doi.org/10.23915/distill.00023, https://distill.pub/2020/growing-ca
23. Parkhi, O.M., Vedaldi, A., Zisserman, A., Jawahar, C.V.: Cats and dogs. In: IEEE Conference on Computer Vision and Pattern Recognition (2012)
24. Pinheiro, P.H.O., Collobert, R.: Recurrent convolutional neural networks for scene labeling. In: Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014. JMLR Workshop and Conference Proceedings, vol. 32, pp. 82–90. JMLR.org (2014), http://proceedings.mlr.press/v32/pinheiro14.html
25. Qin, C., Schlemper, J., Caballero, J., Price, A.N., Hajnal, J.V., Rueckert, D.: Convolutional recurrent neural networks for dynamic MR image reconstruction. IEEE Trans. Med. Imaging **38**(1), 280–290 (2019). https://doi.org/10.1109/TMI.2018.2863670, https://doi.org/10.1109/TMI.2018.2863670
26. Rendell, P.: Turing Universality of the Game of Life, pp. 513–539. Springer London, London (2002), https://doi.org/10.1007/978-1-4471-0129-1_18
27. Sandler, M., Baccash, J., Zhmoginov, A., Howard, A.: Non-discriminative data or weak model? on the relative importance of data and model resolution. In: ICCV Workshop on Real-World Recognition from Low-Quality Images and Videos (2019)
28. Savarese, P.H.P., Maire, M.: Learning implicitly recurrent cnns through parameter sharing. In: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019. OpenReview.net (2019), https://openreview.net/forum?id=rJgYxn09Fm

29. Trigeorgis, G., Snape, P., Nicolaou, M.A., Antonakos, E., Zafeiriou, S.: Mnemonic descent method: A recurrent process applied for end-to-end face alignment. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016. pp. 4177–4187. IEEE Computer Society (2016). https://doi.org/10.1109/CVPR.2016.453, https://doi.org/10.1109/CVPR.2016.453
30. Ulyanov, D., Vedaldi, A., Lempitsky, V.S.: Instance normalization: The missing ingredient for fast stylization. CoRR **abs/1607.08022** (2016), http://arxiv.org/abs/1607.08022
31. Valipour, S., Siam, M., Jägersand, M., Ray, N.: Recurrent fully convolutional networks for video segmentation. In: 2017 IEEE Winter Conference on Applications of Computer Vision, WACV 2017, Santa Rosa, CA, USA, March 24-31, 2017. pp. 29–36. IEEE Computer Society (2017). https://doi.org/10.1109/WACV.2017.11, https://doi.org/10.1109/WACV.2017.11
32. Visin, F., Romero, A., Cho, K., Matteucci, M., Ciccone, M., Kastner, K., Bengio, Y., Courville, A.C.: Reseg: A recurrent neural network-based model for semantic segmentation. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2016, Las Vegas, NV, USA, June 26 - July 1, 2016. pp. 426–433. IEEE Computer Society (2016). https://doi.org/10.1109/CVPRW.2016.60, https://doi.org/10.1109/CVPRW.2016.60
33. Wang, L., Wang, L., Lu, H., Zhang, P., Ruan, X.: Salient object detection with recurrent fully convolutional networks. IEEE Trans. Pattern Anal. Mach. Intell. **41**(7), 1734–1746 (2019). https://doi.org/10.1109/TPAMI.2018.2846598, https://doi.org/10.1109/TPAMI.2018.2846598
34. Wolfram, S.: A New Kind of Science. Wolfram Media Inc., Champaign, Ilinois, USA (2002)
35. Xie, S., Girshick, R.B., Dollár, P., Tu, Z., He, K.: Aggregated residual transformations for deep neural networks. CoRR **abs/1611.05431** (2016), http://arxiv.org/abs/1611.05431
36. Zheng, S., Jayasumana, S., Romera-Paredes, B., Vineet, V., Su, Z., Du, D., Huang, C., Torr, P.H.S.: Conditional random fields as recurrent neural networks. In: 2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015. pp. 1529–1537. IEEE Computer Society (2015). https://doi.org/10.1109/ICCV.2015.179, https://doi.org/10.1109/ICCV.2015.179
37. Zuo, H., Fan, H., Blasch, E., Ling, H.: Combining convolutional and recurrent neural networks for human skin detection. IEEE Signal Process. Lett. **24**(3), 289–293 (2017). https://doi.org/10.1109/LSP.2017.2654803, https://doi.org/10.1109/LSP.2017.2654803

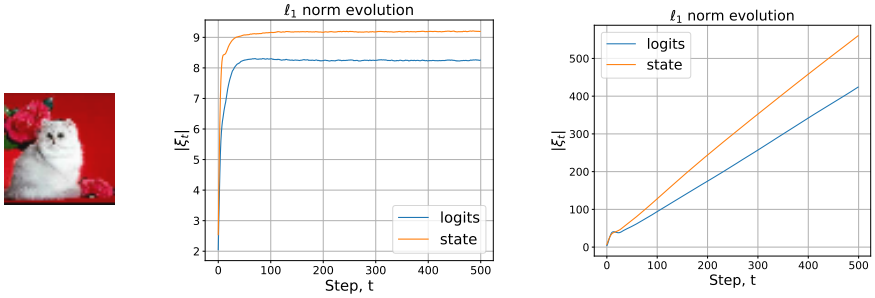
A Appendix

A.1 Empirical differences between resettable and non-resettable models

As discussed in the main text, resettable models produce stable predictions that do not change significantly even if the cellular automaton (CA) is evolved for hundreds of steps beyond the target. It turns out that these two types of models

exhibit other dramatically different behaviors. For example, as shown in Figure 16, ℓ_1 norms of the CA hidden state and model logits saturate in a resettable model with *instance normalization*, but grow linearly with the step number in a non-resettable one. The linear growth in a model with instance normalization is possible due to the presence of residual connections. Norms shown in Figure 16 are normalized to the total number of dimensions in each variable. Looking at the step-by-step changes, we see that fluctuations of the hidden states and logits saturate in amplitude for both models (see Figure 17). However, since the norm of the logits grows linearly in time, final predictions decay in a non-resettable model (while saturating in the resettable CA).

The difference between resettable and non-resettable models is even more striking in models that do not use normalization. As can be seen in Figures 18 and 19, non-resettable models exhibit exponential hidden state growth when unrolled over periods 10× their target unroll. On the other hand, resettable models learn to balance CA hidden state and logit ℓ_1 norms at remarkably stable levels both with and *without* normalization, even though they were not specifically trained for it.

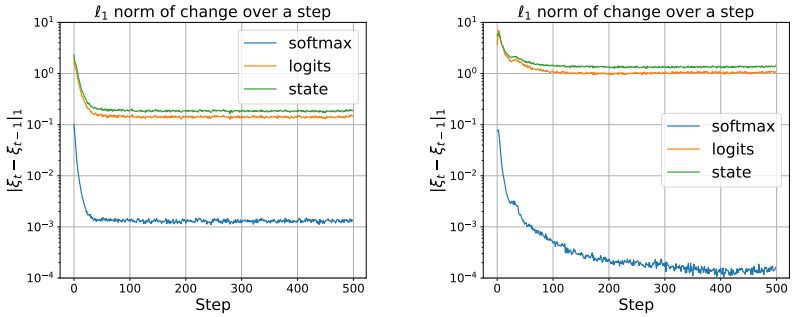


(a) Input (b) Resettable model (c) Non-resettable model

Fig. 16: Evolution of the ℓ_1 norm of the CA state and logits for the input image (a) for models with instance normalization.

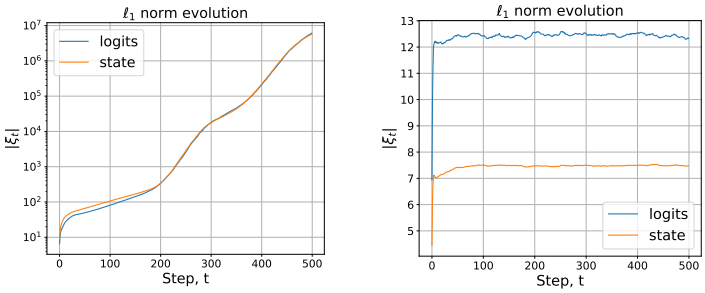
A.2 Adversarial images

Like the vast majority of other deep learning models, our cellular automata are susceptible to adversarial attacks. Images tricking CA into improperly classifying certain regions can be found using a conventional deepdream technique. For example, in Figure 20, we show an image optimized to trick a pre-trained CA into classifying a large image region as pet while it is clear that there is no pet in the image.



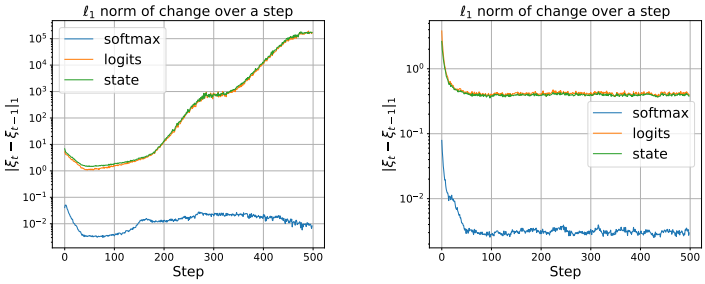
(a) Resettable model (b) Non-resettable model

Fig. 17: Evolution of the ℓ_1 norm of a single-step change for the hidden CA state, logits and final predictions for models with instance normalization. The input image is shown in Figure 16a.



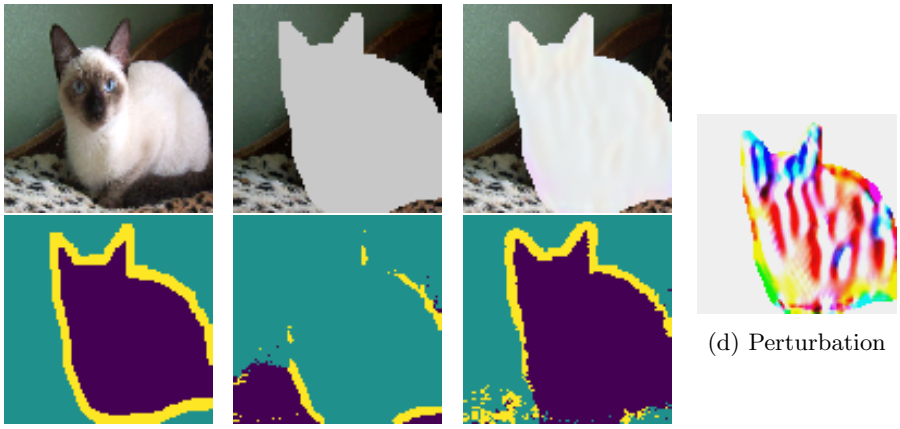
(a) Non-resettable model (b) Resettable model

Fig. 18: Evolution of the ℓ_1 norm of the CA state and logits for models without any normalization. The input image is shown in Figure 16a.



(a) Non-resettable model (b) Resettable model

Fig. 19: Evolution of the ℓ_1 norm of a single-step change of the CA state, logits and predictions for models without normalization. The input image is shown in Figure 16a.



(a) Original (b) Grayed-out (c) Adversarial (d) Perturbation

Fig. 20: *An adversarial image for a pre-trained* 96×96 *resettable CA.* We start with an original image shown in (a) together with the CA prediction. We then cut out a pet region and replace it with a homogeneous gray color as shown in (b). We then use a deepdream technique to find a perturbation localized within the grayed-out pet region that tricks CA into classifying this image region as a "pet". The resulting adversarial image and the CA prediction for it are shown in (c). Notice that by modifying the region inside the animal outline, we can also affect pixel classification outside of it. Image (d) shows an amplified adversarial perturbation.

A.3 High resolution experiments

In our experiments in the main paper on the Oxford Cats and Dogs dataset, we compared cellular automata running on 48×48 and 96×96 images using the same CA state size with the CA processing 96×96 image, but using a 48×48 hidden state resolution. Here we show results of completed runs. Figure 21 shows training curves for these three models. The model acting on a 96×96 image, but using a 48×48 hidden state resolution outperformed all other models with respect to all 3 labels.

(a) "object" (b) "background" (c) "boundary"

Fig. 21: Training curves showing IOU for three labels ("object", "background" and "boundary"). We compare three models: (i) 48×48 image resolution with 48×48 state resolution; (ii) 96×96 image resolution with 96×96 state resolution; (iii) 96×96 image resolution with 48×48 state resolution. Notice that 96×96 model with 48×48 state resolution is about 4 times faster than the model with 96×96 state resolution.