CBMM Memo No. 105 **July 1, 2020**

Do Neural Networks for Segmentation Understand Insideness?

Kimberly Villalobos[*,1], **Vilim Štih**[*,1,2], **Amineh Ahmadinejad**[*,1], **Shobhita Sundaram**[1], **Jamell Dozier**[1], **Andrew Francl**[1], **Frederico Azevedo**[1], **Tomotake Sasaki**[†,3,1], **Xavier Boix**[†,1,•]

[*] and [†] indicate equal contribution
[1] Center for Brains, Minds and Machines, MIT (USA)
[2] Max Planck Institute of Neurobiology (Germany)
[3] Fujitsu Laboratories Ltd. (Japan)
[•] Correspondence to `xboix@mit.edu`

Abstract

The insideness problem is an image segmentation modality that consists of determining which pixels are inside and outside a region. Deep Neural Networks (DNNs) excel in segmentation benchmarks, but it is unclear that they have the ability to solve the insideness problem as it requires evaluating long-range spatial dependencies. In this paper, the insideness problem is analysed in isolation, without texture or semantic cues, such that other aspects of segmentation do not interfere in the analysis. We demonstrate that DNNs for segmentation with few units have sufficient complexity to solve insideness for any curve. Yet, such DNNs have severe problems to learn general solutions. Only recurrent networks trained with small images learn solutions that generalize well to almost any curve. Recurrent networks can decompose the evaluation of long-range dependencies into a sequence of local operations, and learning with small images alleviates the common difficulties of training recurrent networks with a large number of unrolling steps.

This material is based upon work supported by the Center for Brains, Minds and Machines (CBMM), funded by NSF STC award CCF-1231216.

Do Neural Networks for Segmentation Understand Insideness?

Kimberly Villalobos^{*,1}, **Vilim Štih**^{*,1,2}, **Amineh Ahmadinejad**^{*,1},
Shobhita Sundaram^{1}, **Jamell Dozier**^{1}, **Andrew Francl**^{1}, **Frederico Azevedo**^{1},
Tomotake Sasaki^{†,3,1}, **Xavier Boix**$^{\dagger,1,\bullet}$
* and † indicate equal contribution
^{1} Center for Brains, Minds and Machines, MIT (USA)
^{2} Max Planck Institute of Neurobiology (Germany)
^{3} Fujitsu Laboratories Ltd. (Japan)
$^{\bullet}$ Correspondence to `xboix@mit.edu`

Abstract

The insideness problem is an aspect of image segmentation that consists of determining which pixels are inside and outside a region. Deep Neural Networks (DNNs) excel in segmentation benchmarks, but it is unclear if they have the ability to solve the insideness problem as it requires evaluating long-range spatial dependencies. In this paper, the insideness problem is analysed in isolation, without texture or semantic cues, such that other aspects of segmentation do not interfere in the analysis. We demonstrate that DNNs for segmentation with few units have sufficient complexity to solve insideness for any curve. Yet, such DNNs have severe problems with learning general solutions. Only recurrent networks trained with small images learn solutions that generalize well to almost any curve. Recurrent networks can decompose the evaluation of long-range dependencies into a sequence of local operations, and learning with small images alleviates the common difficulties of training recurrent networks with a large number of unrolling steps.

1 Introduction

A key component of image segmentation is to determine whether a pixel is inside or outside a region, *ie.* the "insideness" problem [1, 2]. This problem involves evaluating long-range spatial dependencies. Capturing such long-range dependencies may be challenging for artificial neural networks as pointed out in Minsky & Papert's historic book *Perceptrons* [3] and recent works on capturing other spatial relationships such as containment [4] and connectedness [5].

Deep Neural Networks (DNNs) have been tremendously successful in image segmentation benchmarks, but it is not well understood whether DNNs represent insideness or how. Insideness has been overlooked in DNNs for segmentation since they have been mainly applied to the modality of "semantic segmentation", *ie.* labelling each pixel with its object category [6–12]. The same could be said for DNNs for more advanced segmentation modalities and applications that have been recently introduced, *e.g.* segmentation of individual object instances rather than object categories [13–20] and generating realistic images [21]. In such cases, insideness has not been considered because solutions rely on texture, shape and other visual cues. Yet, investigating whether DNNs understand insideness could reveal new insights about their ability to capture long-range spatial relationships, which is key for a full image understanding.

In this paper, we investigate analytically-derived and learned representations in DNNs for insideness. We take the reductionist approach by isolating insideness such that other components of image segmentation do not provide additional cues and ensure that our analysis focuses on the long-range spatial dependencies involved in insideness. Thus, we analyze the segmentation of Jordan curves, *ie.* closed curves synthetically generated without texture nor object category. We analytically demonstrate that state-of-the-art network architectures, *ie.* DNNs with dilated convolutions [7, 10] and convolutional LSTMs (ConvLSTMs) [22], among other networks, can exactly solve the insideness problem for any curve with network sizes that are easily implemented in practice. The proofs draw on algorithmic ideas from classical work on visual routines [1, 2], namely, the ray-intersection method and the coloring method, to derive equivalent neural networks that implement these algorithms. However, our experiments show that in practice, most DNNs for segmentation do not learn general solutions for insideness, even though the architectures are complex enough to capture the long-range relationships. These DNNs learn to recognize specific features of the family of curves of the training set that do not generalize to new families of curves lacking those features. Only recurrent networks (such as the ConvLSTM) when trained on small images, generalize to almost any given curve of any size. This is because training on small images alleviates the well-known difficulties of training recurrent networks with a large number of unrolling steps [23–26]. It also facilitates learning a strategy that deals with long-range dependencies by breaking them into local operations that are reusable for any curve, even curves in larger images that were not seen during training.

These results add to the growing body of work that demonstrates that DNNs have problems in learning to solve some elemental visual tasks [5, 27–29]. Shalev-Shwartz *et al.* [29] introduced several tasks that DNNs can in theory learn, as demonstrated mathematically, but were unable to solve in practice, even for the training dataset, due to difficulties in the optimization with gradient descent. In contrast, the challenges we report for insideness are related to poor generalization rather than optimization, as our experiments show the networks succeed in solving insideness for the family of curves seen during training. Linsley *et al.* [5] and Kim *et al.* [30] introduced new architectures that better capture the long-range dependencies in images. Here, we show that the training strategy has a significant impact on capturing the long-range dependencies, as even DNNs with the capacity to capture such dependencies do not learn a general solution with standard training strategies. Our results highlight the need to decompose the long-range dependencies in a sequence of local operations, that can be learned with recurrent networks by controlling the number of unrolling steps with the image size.

2 The Reductionist Approach to Insideness

We now introduce a paradigm for analyzing the ability of DNNs to solve insideness. Rather than natural images, we use synthetic stimuli that consist only of one closed curve. In this way, we do not mix the insideness problem with other components of image segmentation found in natural images, *e.g.* discontinuity of segments, representation of the hierarchy of segments, *etc.* This reductionist methodology has the advantage of minimizing the interference of these other factors in analysing abilities of DNNs to capture long-range spatial dependencies. Note that the presence of other factors would obfuscate the specific causes of the network's behavior.

Let $\boldsymbol{X} \in \{0,1\}^{N\times N}$ be an image or a matrix of size $N \times N$ pixels. We use $X_{i,j}$ or $(\boldsymbol{X})_{i,j}$, indistinguishably, to denote the value of the image in position (i,j). We use this notation for indexing elements in any of the images and matrices that appear in the rest of the paper. Also, in the figures we use white and black to represent 0 and 1, respectively.

The *insideness problem* refers to assigning pixels to the inside or the outside of a closed curve. We assume without loss of generality that there is only one closed curve in the image and that it is a digital version of a Jordan curve [31], *ie.* a closed curve without self-crosses nor self-touches and containing only horizontal and vertical turns, as shown in Fig. 2a. We further assume that the curve does not contain the border of the image. The curve is the set of pixels equal to 1 and is denoted by $\mathcal{F}_{\boldsymbol{X}} = \{(i,j)|X_{i,j} = 1\}$.

The pixels in $\boldsymbol{X}$ that are not in $\mathcal{F}_{\boldsymbol{X}}$ can be classified into two categories: the inside and the outside of the curve [31]. We define the segmentation of $\boldsymbol{X}$ as $\boldsymbol{S}(\boldsymbol{X}) \in \{0,1\}^{N\times N}$, where

$$(\boldsymbol{S}(\boldsymbol{X}))_{i,j} = \begin{cases} 0 & \text{if } X_{i,j} \text{ is inside} \\ 1 & \text{if } X_{i,j} \text{ is outside} \end{cases}, \tag{1}$$

and for the pixels in $\mathcal{F}_{\boldsymbol{X}}$, the value of $(\boldsymbol{S}(\boldsymbol{X}))_{i,j}$ can be either 0 or 1. Note that the definition of insideness is rigorously and uniquely determined by the input image itself.

The number of all digital Jordan curves is enormous even if the image size is relatively small, *e.g.* it is more than 10^{47} for the size 32×32 (App. A). In addition, insideness is a global problem; whether a pixel is inside or outside depends on the entire image, and not just on a local area around the pixel. Therefore, simple pattern matching, *ie.* memorization, is impossible in practice.

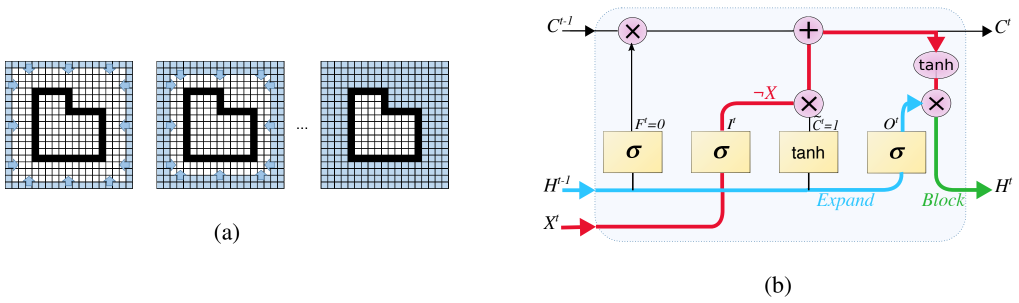


Figure 1: *The Coloring Method with ConvLSTM*. (a) The coloring method consists of several iterations of the *coloring routine*, *ie.* expanding the outside region and blocking it on the curve. (b) Diagram of the ConvLSTM implementing the coloring method, we highlight the connections between layers that are used for insideness. $\neg \boldsymbol{X}$ denotes the element-wise "Boolean not" of $\boldsymbol{X}$.

3 Can DNNs for Segmentation Solve Insideness?

The universal approximation theorem [32] tells us that even a shallow neural network is able to solve the insideness problem. Yet, it could be that the amount of units is too large to be implementable in practice. We show that two commonly used DNN architectures for segmentation are able to perfectly solve the insideness problem, and they are easily implementable in practice. One architecture is feed-forward with dilated convolutions [7, 10] and the other is recurrent, based on ConvLSTMs [8, 22, 33, 34].

Dilated convolutions. Also called atrous convolutions, these are convolutions with upsampled kernels, which enlarge the receptive fields of the units but preserve the number of parameters of the kernel [7, 10]. They facilitate capturing long-range dependencies which are key for segmentation [7, 10]. To demonstrate that there are architectures with dilated convolutions that can solve the insideness problem, we borrow insights from the ray-intersection method. The ray-intersection method [1, 2], also known as the crossings test or the even-odd test [35], is built on the following fact: Any ray that goes from a pixel to the border of the image alternates between inside and outside regions every time it crosses the curve. Therefore, the parity of the total number of such crossings determines the region to which the pixel belongs. If the parity is odd then the pixel is inside, otherwise it is outside. In App. B, we introduce a DNN with dilated convolutions that can implement the ray-intersection algorithm: a network with a number of dilated convolutional layers equal to $\log_2(N)$ (recall N is the image size) with one kernel of 3×3 size, and two convolutional layers with $3N/2$ kernels of 1×1 size. This is the smallest network we could find that solves the insideness problem with dilated convolutions. Larger networks than the one we introduced can also solve the problem, as the network size can be reduced by setting kernels to zero and layers to implement the identity operation.

Convolutional LSTM (ConvLSTM). We now show that a ConvLSTM with just one kernel of size 3×3 is sufficient to solve the insideness problem. Note that applying the ConvLSTM just one step is a local operation that does not tackle long-range dependencies. Thus, the ConvLSTM breaks the long-range dependencies in a sequence of local operations.

Our demonstration is inspired by the coloring method [1, 2], another algorithm for the insideness problem. The algorithm assigns neighbouring pixels to the same region until encountering a border, which is also known as flood filling. We introduce a version of this method that can be implemented as a convLSTM. The coloring method consists of multiple iterations of two steps: *(i)* expand the outside region from the borders of the image (which by assumption are in the outside region) and *(ii)* block the expansion when the curve is reached. The repeated application of these two steps solves the insideness problem, as depicted in Fig. 1a. We call one iteration of expanding and blocking the *coloring routine*.

We use $\boldsymbol{E}^t \in \{0, 1\}^{N \times N}$ (expansion) and $\boldsymbol{B}^t \in \{0, 1\}^{N \times N}$ (blocking) to represent the result of the two operations after iteration t. The *coloring routine* can then be written as *(i)* $\boldsymbol{E}^t = \text{Expand}\left(\boldsymbol{B}^{t-1}\right)$ and *(ii)* $\boldsymbol{B}^t = \text{Block}\left(\boldsymbol{E}^t, \mathcal{F}_{\boldsymbol{X}}\right)$. Let $\boldsymbol{B}^{t-1}$ maintain a value of 1 for all pixels that are known to be outside and 0 for all pixels whose region is not yet determined or belong to the curve. Thus, we initialize $\boldsymbol{B}^0$ to have value 1 (outside) for all border pixels of the image and 0 for the rest. In step *(i)*, the outside region of $\boldsymbol{B}^{t-1}$ is expanded by setting also to 1 (*outside*) its neighboring pixels, and the result is assigned to $\boldsymbol{E}^t$. Next, in step *(ii)*, the pixels in $\boldsymbol{E}^t$ that were labeled with 1 (*outside*) and belong to the curve, $\mathcal{F}_{\boldsymbol{X}}$, are reverted to 0 (*inside*), and the result is assigned to $\boldsymbol{B}^t$. This algorithm ends when the outside region can not expand anymore, which is always less than N^2 iterations (the number of pixels in the image). Therefore, we have $\boldsymbol{E}^{N^2} = \boldsymbol{S}(\boldsymbol{X})$.

In App. C we demonstrate that a ConvLSTM with one kernel applied on an image $\boldsymbol{X}$ can implement the coloring algorithm. In the following we provide a summary of the proof. Let $\boldsymbol{I}^t$, $\boldsymbol{F}^t$, $\boldsymbol{O}^t$, $\boldsymbol{C}^t$, and $\boldsymbol{H}^t \in \mathbb{R}^{N\times N}$ be the activations of the input, forget, and output gates, and cell and hidden states of a ConvLSTM at step t, respectively. By analyzing the equations of the ConvLSTM (equation 11 and equation 12 in App. C) we can see that the output layer, $\boldsymbol{O}^t$, back-projects to the hidden layer, $\boldsymbol{H}^t$. In the coloring algorithm, $\boldsymbol{E}^t$ and $\boldsymbol{B}^t$ are related in a similar manner. Thus, we define $\boldsymbol{O}^t = \boldsymbol{E}^t$ (expansion) and $\boldsymbol{H}^t = \frac{1}{2}\boldsymbol{B}^t$ (blocking). The $\frac{1}{2}$ factor is a technicality due to non-linearities, which is compensated in the output gate and has no relevance in this discussion.

We initialize $\boldsymbol{H}^0 = \frac{1}{2}\boldsymbol{B}^0$ (recall $\boldsymbol{B}^0$ is 1 for all pixels in the border of the image and 0 for the rest). The output gate expands the hidden representations using one 3×3 kernel. To stop the outside region from expanding to the inside of the curve, $\boldsymbol{H}^t$ takes the expansion output $\boldsymbol{O}^t$ and sets the pixels at the curve's location to 0 (inside). This is the same as the element-wise product of $\boldsymbol{O}^t$ and the "Boolean not" of $\boldsymbol{X}$, which is denoted as $\neg\boldsymbol{X}$. Thus, the blocking operation can be implemented as $\boldsymbol{H}^t = \frac{1}{2}(\boldsymbol{O}^t \odot \neg\boldsymbol{X})$, and can be achieved if $\boldsymbol{C}^t$ is equal to $\neg\boldsymbol{X}$. In Fig. 1b we depict these computations.

In App. C we show that the weights of a ConvLSTM with just one kernel of size 3×3 can be configured to reproduce these computations. A key component is that many of the weights use a value that tends to infinity. This value is denoted as q and it is used to saturate the non-linearities of the ConvLSTM, which are hyperbolic tangents and sigmoids. Note that it is common in practice to have weights that asymptotically tend to infinity, *e.g.* when using the cross-entropy loss to train a network [36]. In practice, we found that saturating non-linear units using $q = 100$ is enough to solve the insideness problem for all curves in our datasets. Note that only one kernel is sufficient for ConvLSTM to solve the insideness problem, regardless of image size. Furthermore, networks with multiple stacked ConvLSTM and more than one kernel can implement the coloring method by setting unnecessary ConvLSTMs to implement the identity operation (App. C) and the unnecessary kernels to 0.

Finally, we point out that there are networks with a much lower complexity than LSTMs that can solve the insideness problem, although these networks rarely find applications in practice. In App. D, we show that a convolutional recurrent network as small as having one sigmoidal hidden unit per pixel, with a 3×3 kernel, can also solve the insideness problem for any given curve.

4 Can DNNs for Segmentation Learn Insideness?

After having identified DNNs that with few units have sufficient complexity to solve the insideness problem, we focus on analyzing whether these solutions can be learnt from examples. In the following, we first describe the experimental setup and then analyze the generalization capabilities of the DNNs trained in standard manner.

4.1 Experimental Setup

Recall that the goal of the network is to predict for each pixel in the image whether it is inside or outside of the curve.

Datasets. Given that the number of Jordan curves explodes exponentially with the image size, a procedure that could provide curves without introducing a well-defined bias for learning is unknown. We introduce three algorithms to generate different types of Jordan curves. For each dataset, we generate $95K$ images for training, $5K$ for validation and $10K$ for testing. All the datasets are constructed to fulfill the constraints introduced in Sec. 2. We construct three different datasets of 42×42 pixel image size, called Polar, Spiral and Digs. Fig. 2a, shows examples of curves for each dataset, see App. E for the description on how the curves are generated. Note that the Polar dataset

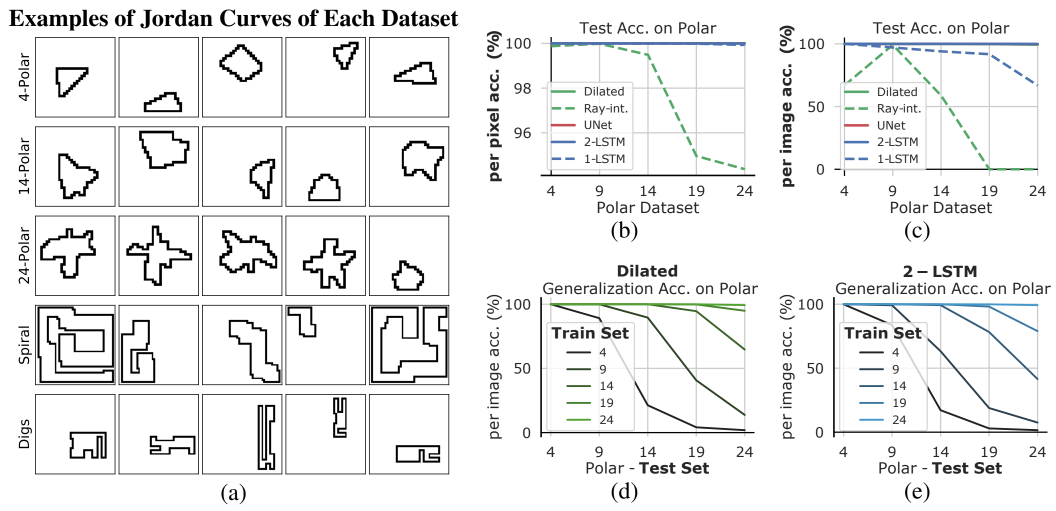


Figure 2: *Datasets and Results in Polar.* (a) Images of the curves used to train and test the DNNs. Each row correspond to a different dataset. (b) and (c) Intra-dataset evaluation using per pixel accuracy and per image accuracy, respectively. (d) and (e) Evaluation using the testing set of each Polar datasets for *Dilated* and *2-LSTM* networks, respectively.

has different complexities depending on the number of vertices of the shape. We refer to each of these datasets as *Polar* with a prefix with the amount of vertices, *e.g.* 24-Polar.

Evaluation metrics. Insideness is evaluated for every pixel except the pixels in the curve $\mathcal{F}_{\boldsymbol{X}}$. We use the following metrics: *per pixel accuracy* (average of the accuracy for inside and outside, evaluated separately, such that it weights the two categories equally as there is an imbalance of inside and outside pixels) and *per image accuracy* (each image is considered correctly classified if all the pixels in the image are correctly classified).

Architectures. We evaluate the networks that we analyzed theoretically and also other common architectures:

- *Feed-forward Architectures:* We use the dilated convolutional DNN (*Dilated*) introduced in Sec. 3. We also evaluate two variants of *Dilated*, which are the Ray-intersection network (*Ray-int.*), which uses a receptive field of $1 \times N$ instead of the dilated convolutions (see App. B), and a convolutional network (*CNN*), which has all the dilation factors set to 1. Finally, we also evaluate *UNet*, which is a popular architecture with skip connections and de-convolutions [6].
- *Recurrent Architectures:* We test the ConvLSTM (*1-LSTM*) corresponding to the architecture introduced in Sec. 3. We initialize the hidden and cell states to 0 (inside) everywhere except the border of the image which is initialized to 1 (outside), such that the network can learn to color by expanding the outside region. We also evaluate a 2-layers ConvLSTM (*2-LSTM*) by stacking one *1-LSTM* after another, both with the initialization of the hidden and cell states of the *1-LSTM*. Finally, to evaluate the effect of such initialization, we test the *2-LSTM* without it (*2-LSTM w/o init.*), *ie.* with the hidden and cell states initialized all to 0. We use backpropagation through time by unrolling 60, 30 or 10 time steps for training (we select the best performing one). For testing, we unroll until there is no change in the output labeling.

Learning. We test a large set of hyperparameters (we trained several thousands of networks per dataset), which we report in detail in App. F. In the following we report the testing accuracy for the hyperparameters that achieved the highest per image accuracy at the validation set.

4.2 Results

We evaluate the DNNs trained in the standard manner, *ie.* using backpropagation with the labeled datasets as described previously, to predict insideness for every pixel.

Intra-dataset Evaluation. In Fig.2b and c we show per pixel and per image accuracy for the networks trained on the same Polar dataset that are being tested. *Dilated*, *2-LSTM* and *UNet* achieve a testing accuracy very close to 100%, but *Ray-int.* and *1-LSTM* perform much worse. Training

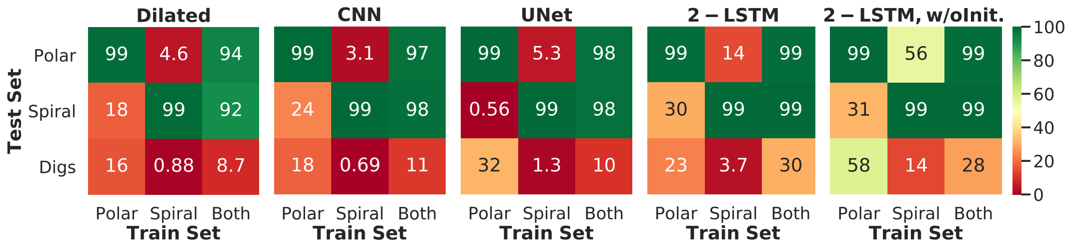


Figure 3: *Cross-dataset Results.* Evaluation of the networks trained in 24-Polar, Spiral and both 24-Polar and Spiral datasets. The tesing sets are 24-Polar, Spiral and Digs datasets.

accuracy of *Ray-int.* and *1-LSTM* is the same as their testing accuracy (Fig. H.7a and b). This indicates an optimization problem similar to the cases reported by [29], as both *Ray-int.* and *1-LSTM* are complex enough to generalize. It is an open question to understand why backpropagation performs so differently in each of these architectures that can all generalize in theory. Finally, note that the per pixel accuracy is in most cases very high, and from now on, we only report the per image accuracy.

Cross-dataset Evaluation. We evaluate if the networks that have achieved very high accuracies (*Dilated*, *2-LSTM* and *UNet*), have learnt the general solution of insideness that we introduced in Sec. 3. To do so, we train on one dataset and test on the different one. In Fig.2d and e, we observe that *Dilated* and *2-LSTM* do not generalize to Polar datasets with larger amount of vertices than the Polar dataset on which they were trained. Only if the networks are trained in 24-Polar, the networks generalize in all the Polar datasets. The same conclusions can be extracted for *UNet* (Fig. H.7c).

We further test generalization capabilities of these networks beyond the Polar dataset. In this more broad analysis, we also include the *CNN* and *2-LSTM w/o init*, by training them on 24-Polar, Spiral and both 24-Polar and Spiral, and test them on 24-Polar, Spiral and Digs separately. We can see in Fig. 3 that all tested networks generalize to new curves of the same family as the training set. Yet, the networks do not generalize to curves of other families. In Fig. H.8, we show qualitative examples of failed segmentations produced by networks trained on 24-Polar and Spiral and tested on the Digs dataset.

Furthermore, note that using a more varied training set ("Both") does not necessarily lead to better cross-dataset accuracy. For example, for *UNet* and *2-LSTM w/o init.*, training on Polar achieves better accuracy in Digs than when training on "Both". Also, for *Dilated*, training on "Both" harms its accuracy: the accuracy drops more than 6% in 24-Polar and Spiral. In this case, the training accuracy is close to 100%, which indicates a problem of overfitting. We tried to address this problem by regularizing using weight decay, but it did not improve the accuracy (App. G). Finally, we also tried fine-tuning to our datasets the state-of-the-art pre-trained networks in segmentation benchmarks (DEXTR [18] and DeepLabv3+ [37]), but they also failed to generalize (App. J).

Understanding the Lack of Generalization. We visualize the networks to understand why the representations learnt do not generalize. In Fig. H.9 and H.10, we analyze different units of *Dilated* trained on 24-Polar and Spiral. We display units of the same kernel from the second, four and sixth layers, by showing the nine images in the testing set that produce the unit to be most active across all images [38]. For each image, we indicate the unit location in the feature map by a green circle. The visualizations suggest that units of the second layer are tuned to local features (*e.g.* Unit 19 is tuned to close parallel lines), while in layer 6 they are tuned to more global ones (*e.g.* Unit 27 captures the space left in the center of a spiral). Thus, the units are tuned to characteristics of the curves in the training set rather than to features that could capture the the long-range dependencies to solve insideness, such as the ones we derived theoretically.

In Fig. H.11, we display the feature maps of *2-LSTM* trained on 24-Polar and Spiral. The figure shows the feature maps of the layers at different time steps. We can see that the network expands the borders of the image, which have been initialized to outside. Yet, some layers expand the curve in both directions, indicating that the blocking operation has not been learned. Also, note that it is impossible to know the direction of expansion from the curve with only local operations, as in one step of the convLSTM. Our analytical solution only expands the borders of the image (see Fig. H.12 for a visualization of the analytical solution).

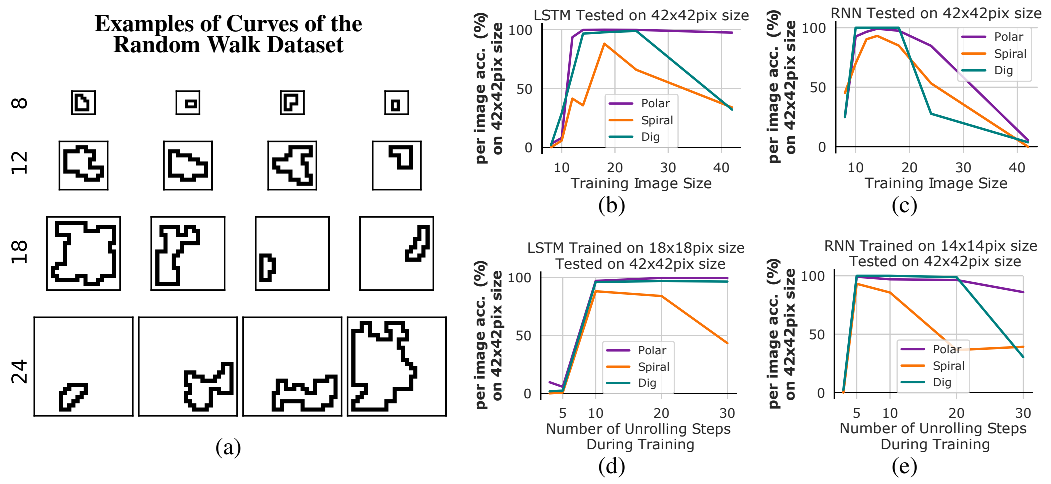


Figure 4: *Datasets and Results of Controling the Number of Unrolling Steps.* (a) Images of the curves of the Random Walk dataset used to train the recurrent networks with different image sizes. Each row correspond to a different image size. (b) and (c) Cross-dataset evaluation (per image accuracy) using the Polar, Spiral and Digs testing sets for *2-LSTM* and *RNN* networks, respectively. (d) and (e) Evaluation of generalization accuracy based on increasing numbers of unrolling steps for *2-LSTM* and *RNN* network, respectively.

5 Learning the *Coloring Routine* with Small Images

Recurrent networks trained by backpropagation through time can be thought of as feed-forward networks with time steps being applied as subsequent layers, *ie.* backpropagation is applied to the "unrolled" version of the recurrent network. Recall that for the insideness problem the prediction error is evaluated after the last unrolling step. Thus, depending on the image size and the shape of the curve a large number of unrolling steps may be required to train the network (*e.g.* the number of unrolling steps for the networks shown in Sec. 3 is bounded by N^2). It is well known that for a large number of unrolling steps backpropagation through time has difficulties capturing long-range relationships [23]. Moreover, the memory requirements are proportional to the number of unrolling steps, making it difficult to scale in practice.

A plethora of attempts have been made to alleviate the problems of learning with a large number of unrolling steps, *e.g.* [24–26]. Here we introduce a simple yet effective strategy for reducing the number of unrolling steps required to learn insideness. Our strategy consists of training the network with small images as they require a smaller number of unrolling steps. To test the network with bigger image sizes, we just unroll it a larger number of steps. Since the recurrent network can learn the *coloring routine* (Sec. 3), it is possible to generalize to larger images and more complex curves. As we have shown before, this routine does not contain long-range operations because it takes into account only 3×3 neighbourhoods; the long-range relationships are captured by applying the *coloring routine* multiple times.

Note that by using small images there is a risk that the dataset lacks enough variability and the network does not generalize beyond the few cases seen during training. Thus, there is a trade-off between the variability in the dataset and the number of unrolling steps required. In the following section, we explore this trade-off and show that adjusting the image size and the number of unrolling steps leads to large gains of generalization accuracy.

5.1 Experimental Setup

The Random Walk Dataset. Polar, Spiral and Digs datasets are hard to adapt to small images without constraining the dataset variability. We introduce a new family of curves which we call the Random Walk dataset. The algorithm to generate the curves is based on a random walk following the connectivity conditions of the Jordan curve we previously introduced. It starts by selecting a random location in the image and at every iteration, choosing with equal probability any of the valid directions (not intersecting the curve constructed so far, or the border). If there are no available directions to expand in, the algorithm backtracks until there are, and continues until the starting point is reached. In Fig. 4a, we show several examples of the curves in images of different sizes.

Architecture. In order to show the generality of our results, we test the *2-LSTM* and another recurrent neural network which we denote as *RNN*. We conducted an architecture search to find a recurrent network that succeeded: a convolutional recurrent neural network with a sigmoidal hidden layer and an output layer that is backwardly connected to the hidden layer. The kernel sizes are 3×3 and 1×1 for the hidden and output layers respectively, with 5 kernels. Observe that this network is sufficiently complex to solve the insideness problem, because it is the network introduced in App. D with an additional layer and connections.

Learning. We train both the *2-LSTM* and the *RNN* on the Random Walk dataset on 7 different image sizes (from 8×8 to 42×42 pixels) and test it with the original images of the 24-Polar, Spiral and Digs datasets (42×42 pixels). To explore the effect of the number of unrolling steps, for each training image size we train the networks using 5, 10, 20, 30 and 60 unrolling steps.

5.2 Results

In Fig. 4b and c, we show the cross-dataset accuracy for the *2-LSTM* and *RNN*, respectively, for different training image sizes. The optimal number of unrolling steps is selected for each training image size. To evaluate the cross-dataset generalization accuracy we apply the trained models to the full size images of the Polar, Spiral, and Digs datasets, using 60 unrolling steps (larger number of unrolling steps does not improve the accuracy). Note that when the networks are trained with small images (14×14 is best for *RNN*, 18×18 is best for *2-LSTM*), at least 80% of the images in all the datasets are correctly segmented. This is a massive improvement of the cross-dataset accuracy compared with the networks trained with large image sizes, as shown in previous section (Fig. 3) and confirmed here again with the Random Walk dataset (less than 40% of the images are correctly segmented in Spiral and Digs datasets, Fig. 4 b).

In Fig. 4d and e, we show the performance of *2-LSTM* and *RNN*, trained with their optimal training image size, when varying the number of unrolling steps. We can observe that both networks generalize with a small number of unrolling steps and fail to generalize as the number of unrolling steps is increased during training. Also, note that as expected, *2-LSTM* is more robust than the *RNN* to large numbers of unrolling steps during training. These positive results demonstrate that training with smaller images and number of unrolling steps is extremely useful in enabling different types of recurrent networks to learn a general solution of insideness across different families of curves not seen during training.

Furthermore, the feature maps of both *2-LSTM* and *RNN* show that the solution implemented by the network is the *Coloring Routine* (Fig. H.13), as the features maps are a non-binary version of the theoretical solution rather than the learned solution with big images (Fig. H.11), which does not generalize because it expands the border of the curve. Note that the *Coloring Routine* emerges in the network and we do not enforce it in any way besides training the network with small images and small number of unrolling steps. Note that we could enforce the network to learn the *Coloring Routine* by providing the ground-truth produced by the routine at each step rather than waiting until the last step. In App. I we show that this per-step strategy also leads to successful cross-dataset generalization accuracy. This result is however less interesting as it requires the per-step ground-truth derived from the analytical solution.

6 Conclusions

We have shown that DNNs with dilated convolutions and convolutional LSTM networks with few units are sufficiently complex to solve the insideness problem for any given curve. Yet, when using the standard training strategies, the units in these networks become specialized to detect characteristics of the curves in the training set and only generalize to curves of the same family as the training. We introduced a strategy to alleviate this limitation by showing that recurrent networks learn the *coloring routine*. The *coloring routine* breaks the evaluation of long-range relationships into local operations and when trained with small images, it generalizes substantially better to new families of curves as it alleviates the well-known difficulties of training recurrent networks with a large number of unrolling steps. We hope that these insights inspire new segmentation training procedures and help future investigations of other important aspects of segmentation beyond insideness (*e.g.* the discontinuity of segments and the hierarchical structure of segments).

Acknowledgements. We are grateful to Tomaso Poggio for his insightful advice and warm encouragement. We also thank Shimon Ullman and Pawan Sinha for helpful feedback and discussions. This work is supported by the Center for Brains, Minds and Machines (funded by NSF STC award CCF-1231216), Fujitsu Laboratories Ltd. (Contract No. 40008401) and the MIT-Sensetime Alliance on Artificial Intelligence.

Statement of Broader Impact. This work provides insights about why DNNs do not generalize well to segmenting families of curves that were not included in the training set, and investigates the representations that lead to general solutions and how to learn them. We contribute to establishing the reductionist approach to understanding DNNs, where specific problem components are investigated by minimizing the interference of other factors. The reductionist approach allows to open the "black box" of DNNs and analyze precisely how DNNs represent specific aspects of visual understanding.

Furthermore, we add to the body of work on DNN's failure modes by demonstrating a potential vulnerability with the insideness problem. This creates a risk of pointing out vulnerabilities that can be exploited with negative societal consequences [39]; for instance, in the case of visual tasks, demonstrating ways to fool surveillance or self-driving cars [40]. However, understanding specific vulnerabilities is crucial towards building more robust, explainable networks.

References

[1] Shimon Ullman. Visual routines. *Cognition*, 1984.

[2] Shimon Ullman. *High-Level Vision: Object Recognition and Visual Cognition*. MIT Press, 1st edition, 1996.

[3] Marvin L. Minsky and Seymour A. Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, 1st edition, 1969.

[4] Junkyung Kim, Matthew Ricci, and Thomas Serre. Not-so-clevr: learning same–different relations strains feedforward neural networks. *Interface focus*, 2018.

[5] Drew Linsley, Junkyung Kim, Vijay Veerabadran, Charles Windolf, and Thomas Serre. Learning long-range spatial dependencies with horizontal gated recurrent units. In *NeurIPS*, 2018.

[6] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*, 2015.

[7] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. In *ICLR*, 2016.

[8] Francesco Visin, Marco Ciccone, Adriana Romero, Kyle Kastner, Kyunghyun Cho, Yoshua Bengio, Matteo Matteucci, and Aaron Courville. ReSeg: A recurrent neural network-based model for semantic segmentation. In *CVPR Workshops*, 2016.

[9] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. SegNet: A deep convolutional encoder-decoder architecture for image segmentation. *TPAMI*, 2017.

[10] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. DeepLab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs. *TPAMI*, 2018.

[11] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015.

[12] Fahad Lateef and Yassine Ruichek. Survey on semantic segmentation using deep learning techniques. *Neurocomputing*, 2019.

[13] Ke Li, Bharath Hariharan, and Jitendra Malik. Iterative instance segmentation. In *CVPR*, 2016.

[14] Yi Li, Haozhi Qi, Jifeng Dai, Xiangyang Ji, and Yichen Wei. Fully convolutional instance-aware semantic segmentation. In *CVPR*, 2017.

[15] Gwangmo Song, Heesoo Myeong, and Kyoung Mu Lee. SeedNet: Automatic seed generation with deep reinforcement learning for robust interactive segmentation. In *CVPR*, 2018.

[16] Liang-Chieh Chen, Alexander Hermans, George Papandreou, Florian Schroff, Peng Wang, and Hartwig Adam. MaskLab: Instance segmentation by refining object detection with semantic and direction features. In *CVPR*, 2018.

[17] Ronghang Hu, Piotr Dollár, Kaiming He, Trevor Darrell, and Ross Girshick. Learning to segment every thing. In *CVPR*, 2018.

[18] Kevis-Kokitsi Maninis, Sergi Caelles, Jordi Pont-Tuset, and Luc Van Gool. Deep extreme cut: From extreme points to object segmentation. In *CVPR*, 2018.

[19] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation. In *CVPR*, 2018.

[20] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. In *ICCV*, 2017.

[21] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *CVPR*, 2017.

[22] SHI Xingjian, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In *NIPS*, 2015.

[23] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 1994.

[24] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[25] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *ICML*, 2013.

[26] Audrunas Gruslys, Rémi Munos, Ivo Danihelka, Marc Lanctot, and Alex Graves. Memory-efficient backpropagation through time. In *NIPS*, 2016.

[27] Rosanne Liu, Joel Lehman, Piero Molino, Felipe Petroski Such, Eric Frank, Alex Sergeev, and Jason Yosinski. An intriguing failing of convolutional neural networks and the coordconv solution. In *NeurIPS*, 2018.

[28] Xiaolin Wu, Xi Zhang, and Xiao Shu. Cognitive deficit of deep learning in numerosity. In *AAAI*, 2018.

[29] Shai Shalev-Shwartz, Ohad Shamir, and Shaked Shammah. Failures of gradient-based deep learning. In *ICML*, 2017.

[30] Junkyung Kim, Drew Linsley, Kalpit Thakkar, and Thomas Serre. Disentangling neural mechanisms for perceptual grouping. In *ICLR*, 2020.

[31] T. Yung Kong. Digital topology. In Larry S. Davis, editor, *Foundations of Image Understanding*, pages 73–93. Springer, 2001.

[32] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[33] Ruiyu Li, Kaican Li, Yi-Chun Kuo, Michelle Shu, Xiaojuan Qi, Xiaoyong Shen, and Jiaya Jia. Referring image segmentation via recurrent refinement networks. In *CVPR*, 2018.

[34] Md Zahangir Alom, Mahmudul Hasan, Chris Yakopcic, Tarek M Taha, and Vijayan K Asari. Recurrent residual convolutional neural network based on u-net (r2u-net) for medical image segmentation. *arXiv preprint arXiv:1802.06955*, 2018.

[35] Eric Haines. Point in polygon strategies. In Paul Heckbert, editor, *Graphics Gems IV*, pages 24–46. Academic Press, 1994.

[36] Daniel Soudry, Elad Hoffer, Mor Shpigel Nacson, Suriya Gunasekar, and Nathan Srebro. The implicit bias of gradient descent on separable data. *JMLR*, 2018.

[37] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. *ECCV*, 2018.

[38] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *ECCV*, 2014.

[39] D Heaven. Why deep-learning ais are so easy to fool. *Nature*, 574(7777):163, 2019.

[40] Marcus Comiter. Attacking artificial intelligence: AI's security vulnerability and what policymakers can do about it. Belfer Center for Science and International Affairs, Harvard Kennedy School, August 2019.

[41] Azriel Rosenfeld. Connectivity in digital pictures. *J. ACM*, 17(1):146–160, 1970.

[42] Frank Harary. *Graph Theory*. Addison-Wesley, 1969.

[43] Hiroaki Iwashita, Yoshio Nakazawa, Jun Kawahara, Takeaki Uno, and Shinichi Minato. Fast computation of the number of paths in a grid graph. In *The 16th Japan Conference on Discrete and Computational Geometry and Graphs (JCDCG2 2013)*, Tokyo, September 2013.

[44] A140517: Number of cycles in an n x n grid. In *The On-Line Encyclopedia of Integer Sequences*. [Online]. Available: `https://oeis.org/A140517`.

[45] Artem M. Karavaev and Hiroaki and Iwashita. Table of n, a(n) for n = 0..26. In *The On-Line Encyclopedia of Integer Sequences*. [Online]. Available: `https://oeis.org/A140517/b140517.txt`.

[46] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, 2010.

[47] Mark Everingham, Luc van Gool, Chris Williams, John Winn, and Andrew Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. `http://host.robots.ox.ac.uk/pascal/VOC/voc2012/`.

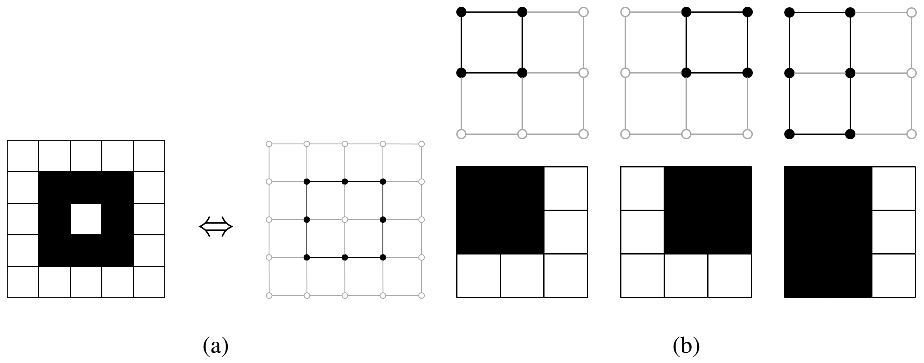


Figure A.1: *Subgraph Representations of Figures.* (a) A figure in an image of size 5 $\times$ 5 pixels (left) and its subgraph representation in a grid graph of 5 $\times$ 5 vertices (right). (b) Cycles that are not digital Jordan curves (top) and their correspondents (bottom).

A Number of Digital Jordan Curves

We now introduce a procedure to derive a lower bound of the number of Jordan curves in an image. We represent an image of size $N \times N$ pixels by using a grid graph (square lattice) with $N \times N$ vertices. We employ 4-adjacency for black pixels and corresponding grid points, and 8-adjacency for white pixels and their counterpart. Then, a curve (the set of black pixels) corresponds to a subgraph of the base grid graph (Fig. A.1a).

In this representation, a digital Jordan curve is defined as a subgraph specified by a sequence of vertices $(\mathrm{v}_0, \mathrm{v}_1, \ldots, \mathrm{v}_L)$ satisfying the following conditions [31, 41]:

1. v_r is 4-adjacent to v_s if and only if $r \equiv s \pm 1 \pmod{L+1}$,
2. $\mathrm{v}_r = \mathrm{v}_s$ if and only if $r = s$, and
3. $L \geq 4$.

The “if” part of condition 1 means that the black pixels lie consecutively and the curve formed by the black pixels is closed. The “only if” part of conditions 1 and 2 assures that the curve never crosses or touches itself. The condition 3 excludes exceptional cases which satisfy conditions 2 and 3 but cannot be considered as digital versions of the Jordan curve (see [41] for details). Note that any digital Jordan curve is a cycle [42] in a grid graph but not vice versa. Figure A.1b shows examples of cycles that are not digital Jordan curves.

The numbers of all cycles in grid graphs of different sizes were computed up to 27$\times$27 vertices [43,44], and we utilize this result to get lower bounds for the number of digital Jordan curves with the following considerations.

Although a cycle in a grid graph is not necessarily a digital Jordan curve as shown above, we can obtain a digital Jordan curve in a larger image from any cycle by “upsampling" as shown in Fig. A.2a. Note that there are other digital Jordan curves than the ones obtained in this manner (therefore we get a lower bound with this technique). See Fig. A.2b for examples.

We also consider “padding" shown in Fig. A.3 to assure that a digital Jordan curve does not contain the border of a image (this is what we assume in the main body of the paper).

Taking everything into consideration, we can obtain a lower bound of the number of digital Jordan curves in an $N \times N$ image that does not contain border pixels utilizing the above-mentioned result [43, 44], upsampling and padding. Table 1 shows lower bounds obtained in this way. For example, starting with the row 2 of [45] in [44] (this represents the number of all cycles in the grid graph with 3 $\times$ 3 vertices), we get a lower bound 13 for the number of digital Jordan curves in 5 $\times$ 5 images by considering the upsampling and get the same number as a lower bound for the number of digital Jordan curves that do not contain border pixels in 7 $\times$ 7 images by considering the padding.

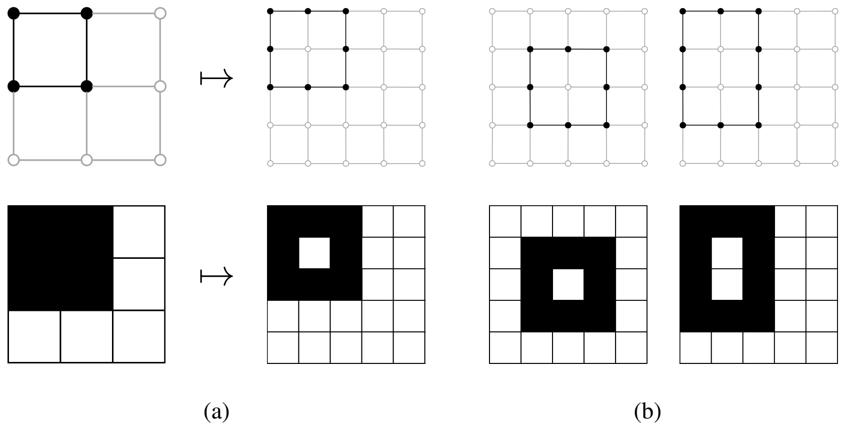


Figure A.2: *"Upsampling" Operation and Its Limitations.* (a) Depiction of "upsampling" operation. (b) Digital Jordan curves that cannot be obtained by the upsampling shown in (a). (Left) The issue is the place of the digital Jordan curve. We can get the same curve on the upper-left, upper-right, lower-left and lower-right corners but cannot get the one in the center. (Right) The issue is the length of the side. We cannot get a side with 4 vertices (4 pixels) nor with even number vertices (pixels) in general.

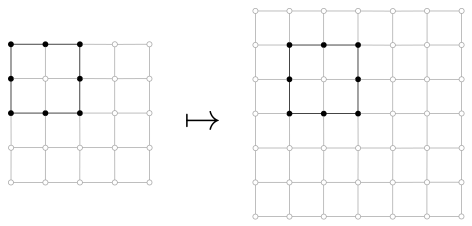


Figure A.3: *Depiction of "Padding".*

B Implementing the Ray-intersection algorithm with Dilated Convolutional Networks

Dilated convolutions facilitate capturing long-range dependencies which are key for segmentation [7, 10]. To demonstrate that there are architectures with dilated convolutions that can solve the insideness problem, we borrow insights from the ray-intersection method. The ray-intersection method [1,2], also known as the crossings test or the even-odd test [35], is built on the following fact: Any ray that goes from a pixel to the border of the image alternates between inside and outside regions every time it crosses the curve. Therefore, the parity of the total number of such crossings determines the region to which the pixel belongs. If the parity is odd then the pixel is inside, otherwise it is outside (see Fig. B.4a).

The definition of a crossing should take into account cases like the one depicted in Fig. B.4b, in which the ray intersects the curve, but does not change region after the intersection. To address these cases, we enumerate all possible intersections of a ray and a curve, and analyze which cases should count as crossings and which ones should not. Without loss of generality, we consider only horizontal rays. As we can see in Fig. B.4c, there are only five cases for how a horizontal ray can intersect the curve. The three cases at the top of Fig. B.4c, are crosses because the ray goes from one region to the opposite one, while the two cases at the bottom (like in Fig. B.4b) are not considered crosses because the ray remains in the same region.

Let $\vec{X}(i,j) \in \{0,1\}^{1\times N}$ be a horizontal ray starting from pixel (i,j), which we define as

$$\vec{X}(i,j) = [X_{i,j}, X_{i,j+1}, X_{i,j+2}, \dots, X_{i,N}, 0, \dots, 0], \tag{2}$$

where zeros are padded to the vector if the ray goes outside the image, such that $\vec{X}(i,j)$ is always of dimension N. Let $\vec{X}(i,j) \cdot \vec{X}(i+1,j)$ be the inner product of the ray starting from (i,j) and the

Table 1: *Lower bounds (LBs) of the number of digital Jordan curves in $N \times N$ images that do not contain border pixels.*

N	5	7	9	$\cdots$	31	33	35	$\cdots$	55
LB	1	13	213	$\cdots$	1.203×10^{47}	1.157×10^{54}	3.395×10^{61}	$\cdots$	6.71×10^{162}

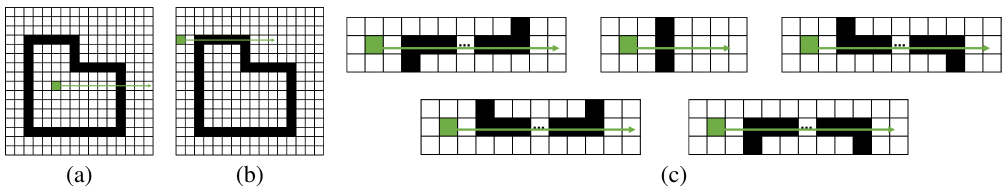


Figure B.4: *Intersections of the Ray and the Curve.* (a) Example of ray going from one region to the opposite one when crossing the curve. (b) Example of ray staying in the same region after intersecting the curve. (c) All cases in which a ray could intersect a curve. In the three cases above the ray travels from one region to the opposite one, while in the two cases below the ray does not change regions.

ray starting from the pixel below, $(i+1,j)$. Note that the contribution to this inner product from the three cases at the top of Fig. B.4c (the crossings) is odd, whereas the contribution from the other two intersections is even. Thus, the parity of $\vec{X}(i,j) \cdot \vec{X}(i+1,j)$ is the same as the parity of the total number of crosses and determines the insideness of the pixel (i,j), *ie.*

$$(\boldsymbol{S}(\boldsymbol{X}))_{i,j} = \text{parity}\left(\vec{X}(i,j) \cdot \vec{X}(i+1,j)\right). \tag{3}$$

In the following we prove that equation 3 can be implemented with a neural network with dilated convolutions. The demonstration is based on implementing the dot product in equation 3 with multiple layers of dilated convolutions, as they enable capturing the information across the ray, and then, calculating the parity with another neural network. We first introduce a feed-forward convolutional DNN for which there exist parameters that reproduce equation 3. Then, we show that one of the layers in this network can be better expressed with multiple dilated convolutions. Finally, we introduce the network to calculate the parity.

B.1 Overview of the Network to Implement the Ray-Intersection Method

The smallest CNN that we found that implements the ray-intersection method has 4-layers. Fig. B.5a depicts the architecutre. As we show in the following, the first two layers compute $\vec{X}(i,j)\cdot\vec{X}(i+1,j)$, and the last two layers compute the parity. We use $\boldsymbol{H}^{(k)} \in \mathbb{R}^{N\times N}$ to denote the activations of the units at the k-th layer. These are obtained after applying the *ReLU* activation function, which is denoted as $[\,]_+$.

First and Second Layer: Inner product. For the sake of simplicity, we only use horizontal rays, but the network that we introduce can be easily adapted to any ray direction. The first layer implements all products needed for the inner products across all rays in the image, *ie.* $X_{i,j} \cdot X_{i+1,j}, \forall(i,j)$. Note that there is exactly one product per pixel, and each product can be reused for multiple rays. For convenience, $H^{(1)}_{i,j}$ represents the product in pixel (i,j), *ie.* $H^{(1)}_{i,j} = X_{i,j} \cdot X_{i+1,j}$. Since the input consists of binary images, each product can be reformulated as

$$H^{(1)}_{i,j} = \begin{cases} 1 & \text{if} \quad X_{i,j} = X_{i+1,j} = 1 \\ 0 & \text{otherwise} \end{cases}. \tag{4}$$

This equality can be implemented with a *ReLU*: $H^{(1)}_{i,j} = [1 \cdot X_{i,j} + 1 \cdot X_{i+1,j} - 1]_+$. Thus, $\boldsymbol{H}^{(1)}$ is a convolutional layer with a 2×1 kernel that detects the intersections shown in Fig. B.4c. This layer can also be implemented with a standard convolutional layer with a 3×3 kernel, by setting the unnecessary elements of the kernel to 0.

The second layer sums over the products of each ray. To do so, we use a kernel of dimension $1 \times N$ with weights equal to 1 and bias equal to 0, *ie.* $H^{(2)}_{i,j} = \boldsymbol{1}_{1\times N} \cdot H^{(1)}_{i,j} = \vec{X}(i,j) \cdot \vec{X}(i+1,j)$, in

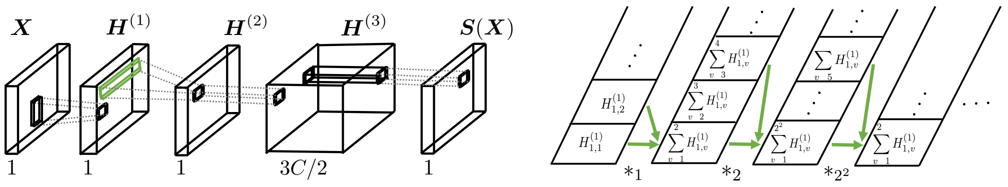


Figure B.5: *Dilated Convolutions from the Ray-Intersection Method.* (a) The receptive field colored in green has size $1 \times N$, and it can be substituted by an equivalent network composed of multiple dilated convolutions. (b) The $1 \times N$ kernel of the ray-intersection network is equivalent to multiple dilated convolutional layers. The figure shows an horizontal ray of activations from different layers, starting from the first layer $H^{(1)}$. The green arrows indicate the locations in the ray that lead to the desired sum of activations to implement the $1 \times N$ kernel, *ie.* the sum of the ray.

which $\boldsymbol{1}_{I\times J}$ denotes the matrix of size $I \times J$ with all entries equal to 1. Zero-padding is used to keep the kernel size constant across the image.

Note that the shape of the kernel, $1 \times N$, is not common in the DNN literature. Here, it is necessary to capture the long-range dependencies of insideness. In App.B.2, we show that the $1 \times N$ kernel can be substituted by multiple layers of dilated convolutions.

Third and Fourth Layers: Parity. To calculate the parity of each unit's value in $\boldsymbol{H}^{(2)}$, we borrow the DNN introduced by [29] (namely, Lemma 3 in the supplemental material of the paper). This network obtains the parity of any integer bounded by a constant C. The network has $3C/2$ hidden *ReLUs* and one output unit, which is 1 if the input is even, 0 otherwise (see App. B.3 for details).

We apply this parity network to all units in $\boldsymbol{H}^{(2)}$ via convolutions, reproducing the network for each unit. Since a ray through a closed curve in an $N \times N$ image can not have more than N crossings, C is upper bounded by N. Thus, the third layer has $3N/2$ kernels, and both the third and output layer are convolutions with a 1×1 kernel. At this point we have shown that the DNN explained above is feasible in practice, as the number of kernels is $O(N)$, and it requires no more than 4 convolutional layers with *ReLUs*. The network has a layer with a kernel of size $1 \times N$, and next we show that this layer is equivalent to several layers of dilated convolutions of kernel size 3×3.

B.2 Dilated Convolutions to Implement the $1 \times N$ kernel

We use $*_d$ to denote a dilated convolution, in which d is the dilation factor. Let $\boldsymbol{H} \in \mathbb{R}^{N\times N}$ be the units of a layer and let $\boldsymbol{K} \in \mathbb{R}^{k\times k}$ be a kernel of size $k \times k$. A dilated convolution is defined as follows: $(\boldsymbol{H} *_d \boldsymbol{K})_{i,j} = \sum_{-\lfloor k/2\rfloor \le v,w \le \lfloor k/2\rfloor} H_{i+dv,j+dw} \cdot K_{v,w}$, in which $H_{i+dv,j+dw}$ is 0 if $i + dv$ or $j + dw$ are smaller than 0 or larger than N, *ie.* we abuse notation for the zero-padding. Note that in the dilated convolution the kernel is applied in a sparse manner, every d units, rather than in consecutive units. See [7, 10] for more details on dilated convolutions.

Recall the kernel of size $1 \times N$ is set to $\boldsymbol{1}_{1\times N}$ so as to perform the sum of the units corresponding to the ray in the first layer, *ie.* $\sum_{0\le v<N} H^{(1)}_{i,j+v}$. We can obtain this long-range sum with a series of dilated convolutions using the following 3×3 kernel:

$$\boldsymbol{K} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}. \tag{5}$$

First, we apply this $\boldsymbol{K}$ to the $\boldsymbol{H}^{(1)}$ through $*_1$ in order to accumulate the first two entries in the ray, which yields: $\left(\boldsymbol{H}^{(2)}\right)_{i,j} = \left(\boldsymbol{H}^{(1)} *_1 \boldsymbol{K}\right)_{i,j} = \sum_{0\le v\le 1} H^{(1)}_{i,j+v}$. As shown in Fig. B.5b, to accumulate the next entries of the ray, we can apply $\boldsymbol{K}$ with a dilated convolution of dilation factor $d = 2$, which leads to $\left(\boldsymbol{H}^{(3)}\right)_{i,j} = \sum_{0\le v<4} H^{(1)}_{i,j+v}$. To further accumulate more entries of the ray, we need larger dilation factors. It can be seen in Fig. B.5b that these dilation factors are powers of 2, which yield the following expression:

$$\left(\boldsymbol{H}^{(l)}\right)_{i,j} = \left(\boldsymbol{H}^{(l-1)} *_{2^{l-2}} \boldsymbol{K}\right)_{i,j} = \sum_{0\leq v<2^{l-2}} H^{(1)}_{i,j+v}. \tag{6}$$

Observe that when we reach layer $l = \log_2(N) + 2$, the units accumulate the entire ray of length N, *ie.* $\sum_{0\leq v<N} H^{(1)}_{i,j+v}$. Networks with dilation factors $d = 2^l$ are common in practice, *e.g.* [7] uses these exact dilation factors.

In summary, DNNs with dilated convolutions can solve the insideness problem and are implementable in practice, since the number of layers and the number of kernels grow logarithmically and linearly with the image size, respectively.

B.3 Parity Network by [29]

To calculate the parity of each unit's value in $\boldsymbol{H}^{(2)}$, we borrow the DNN introduced by Shalev-Shwartz *et al.* (namely, Lemma 3 in the supplemental material of [29]). This network obtains the parity of any integer bounded by a constant C. The network has $\frac{3C}{2}$ hidden units with *ReLUs* and one output unit, which is 1 if the input is even, 0 otherwise. Since such a network requires an upper bound on the number whose parity is being found, we define C as the maximum number of times that a horizontal ray can cross $\mathcal{F}_{\boldsymbol{X}}$. This number can be regarded as an index to express the complexity of the shape.

There is a subtle difference between the network introduced by [29] and the network we use in the paper. In [29], the input of the network is a string of bits, but in our case, the sum is done in the previous layer, through the dilated convolutions. Thus, we use the network in [29] after the sum of bits is done, *ie.* after the first dot product in the first layer in [29].

To calculate the parity, for each even number between 0 and C (0 included), $\{2i \mid 0 \leq i \leq \lfloor C/2 \rfloor\}$, the network has three hidden units that threshold at $(2i - \frac{1}{2})$, $2i$ and $(2i + \frac{1}{2})$, *ie.* $-0.5, 0, 0.5, 1.5, 2, 2.5, 3.5, 4, 4.5, \ldots$ The output layer linearly combines all the hidden units and weights each triplet of units by 2, -4 and 2. Observe that when the input is an odd number, the three units in the triplet are either all below or all above the threshold. The triplets that are all below the threshold contribute 0 to the output because the units are inactive, and the triplets that are all above the threshold also contribute 0 because the linear combination is $2(2i - \frac{1}{2}) - 4(2i) + 2(2i + \frac{1}{2}) = 0$. For even numbers, the triplet corresponding to that even number has one unit below, equal and above the threshold. The unit that is above the threshold contributes 1 to the output, yielding the parity function.

C Implementing the *Coloring Routine* with a Convolutional LSTM

Here we prove that a ConvLSTM can implement the coloring routine, namely, the iteration of the expansion and the blocking operations. A ConvLSTM applied on an image $\boldsymbol{X}$ is defined as the following set of layers (see [22] for a comprehensive introduction to the ConvLSTM):

$$\boldsymbol{I}^t = \sigma\left(\boldsymbol{W}^{xi} * \boldsymbol{X} + \boldsymbol{W}^{hi} * \boldsymbol{H}^{t-1} + \boldsymbol{b}^i\right), \tag{7}$$

$$\boldsymbol{F}^t = \sigma\left(\boldsymbol{W}^{xf} * \boldsymbol{X} + \boldsymbol{W}^{hf} * \boldsymbol{H}^{t-1} + \boldsymbol{b}^f\right), \tag{8}$$

$$\tilde{\boldsymbol{C}}^t = \tanh\left(\boldsymbol{W}^{xc} * \boldsymbol{X} + \boldsymbol{W}^{hc} * \boldsymbol{H}^{t-1} + \boldsymbol{b}^c\right), \tag{9}$$

$$\boldsymbol{C}^t = \boldsymbol{F}^t \odot \boldsymbol{C}^{t-1} + \boldsymbol{I}^t \odot \tilde{\boldsymbol{C}}^t, \tag{10}$$

$$\boldsymbol{O}^t = \sigma\left(\boldsymbol{W}^{xo} * \boldsymbol{X} + \boldsymbol{W}^{ho} * \boldsymbol{H}^{t-1} + \boldsymbol{b}^o\right), \tag{11}$$

$$\boldsymbol{H}^t = \boldsymbol{O}^t \odot \tanh\left(\boldsymbol{C}^t\right), \tag{12}$$

where $\boldsymbol{I}^t$, $\boldsymbol{F}^t$, $\boldsymbol{C}^t$, $\boldsymbol{O}^t$ and $\boldsymbol{H}^t \in \mathbb{R}^{N\times N}$ are the activation of the units of the input, forget, cell state, output and hidden layers at t, respectively. Note that $\boldsymbol{C}^t$ has been decomposed with the help of the auxiliary equation defining $\tilde{\boldsymbol{C}}^t$. Note also that each of these layers use a different set of weights that are applied to $\boldsymbol{X}$ and to $\boldsymbol{H}^t$ denoted as $\boldsymbol{W} \in \mathbb{R}^{N\times N}$ with superindices that indicate the connections between layers, *e.g.* $\boldsymbol{W}^{xi}$ are the weights that connect $\boldsymbol{X}$ to $\boldsymbol{I}$. Similarly, the biases are denoted as $\boldsymbol{b} \in \mathbb{R}^{N\times N}$ with the superindices indicating the layers. The symbols $*$ and $\odot$ denote the (usual, not dilated) convolution and the element-wise product, respectively. Finally, σ and $\tanh$ are the sigmoid and the hyperbolic tangent, which are used as non-linearities.

We can see by analyzing equation 11 and equation 12 that the output layer, $\boldsymbol{O}^t$, back-projects to the hidden layer, $\boldsymbol{H}^t$. In the coloring algorithm, $\boldsymbol{E}^t$ and $\boldsymbol{B}^t$ are related in a similar manner. Thus, we define $\boldsymbol{O}^t = \boldsymbol{E}^t$ (expansion) and $\boldsymbol{H}^t = \frac{1}{2}\boldsymbol{B}^t$ (blocking), as depicted in Fig. 1b. The $\frac{1}{2}$ factor will become clear below, and it does not affect the correctness. We initialize $\boldsymbol{H}^0 = \frac{1}{2}\boldsymbol{B}^0$ (recall $\boldsymbol{B}^0$ is 1 for all pixels in the border of the image and 0 for the rest). We now show how to implement the iteration of the expansion and the blocking operations with the ConvLSTM:

(i) Expansion, $\boldsymbol{O}^t$: We set the output layer in equation 11 in the following way:

$$\boldsymbol{O}^t = \sigma\left(2q\boldsymbol{1}_{3\times 3} * \boldsymbol{H}^{t-1} - \frac{q}{2}\boldsymbol{1}_{N\times N}\right). \tag{13}$$

Note that this layer does not use the input, and sets the convolutional layer $\boldsymbol{W}^{ho}$ to use a 3×3 kernel that is equal to $2q\boldsymbol{1}_{3\times 3}$, in which q is a scalar constant, and the bias equal to $-\frac{q}{2}\boldsymbol{1}_{N\times N}$. For very large values of q, this layer expands the outside region. This can be seen by noticing that for a unit in $\boldsymbol{H}^{t-1}$, if at least one neighbor has value $1/2$, then $O^t_{i,j} = \lim_{q\to\infty}\sigma(q) = 1$. Also, when all neighbouring elements of the unit are 0, then no expansion occurs because $O^t_{i,j} = \lim_{q\to\infty}\sigma(-\frac{q}{2}) = 0$.

(ii) Blocking, $\boldsymbol{H}^t$: To stop the outside region from expanding to the inside of the curve, $\boldsymbol{H}^t$ takes the expansion output $\boldsymbol{O}^t$ and sets the pixels at the curve's location to 0 (inside). This is the same as the element-wise product between $\boldsymbol{O}^t$ and the element-wise "Boolean not" of $\boldsymbol{X}$, which is denoted as $\neg\boldsymbol{X}$. Thus, the blocking operation can be implemented as $\boldsymbol{H}^t = \frac{1}{2}(\boldsymbol{O}^t \odot \neg\boldsymbol{X})$. Observe that if $\boldsymbol{C}^t = \neg\boldsymbol{X}$, this is equal to equation 12 of the LSTM, because $\tanh(0) = 0$ and $\tanh(1) = 1/2$, *ie.*

$$\boldsymbol{H}^t = \boldsymbol{O}^t \odot \tanh\left(\boldsymbol{C}^t\right) = \frac{1}{2}\boldsymbol{O}^t \odot \neg\boldsymbol{X}. \tag{14}$$

We can obtain $\boldsymbol{C}^t = \neg\boldsymbol{X}$, by imposing $\boldsymbol{I}^t = \neg\boldsymbol{X}$ and $\boldsymbol{C}^t = \boldsymbol{I}^t$, as shown in Fig. 1b. To do so, let $\boldsymbol{W}^{xi} = -q\boldsymbol{1}_{1\times 1}$, $\boldsymbol{W}^{hi} = \boldsymbol{0}_{N\times N}$, and $\boldsymbol{b}^i = \frac{q}{2}\boldsymbol{1}_{N\times N}$, and equation 8 becomes the following expression:

$$\boldsymbol{I}^t = \lim_{q\to\infty}\sigma\left(-q\boldsymbol{1}_{1\times 1} * \boldsymbol{X} + \frac{q}{2}\boldsymbol{1}_{N\times N}\right). \tag{15}$$

Observe that when q tends to infinity, we have $I^t_{i,j} = \lim_{q\to\infty}\sigma(\frac{q}{2}) = 1$ when $X_{i,j} = 0$ and $I^t_{i,j} = \lim_{q\to\infty}\sigma(-\frac{q}{2}) = 0$ when $X_{i,j} = 1$, which means $\boldsymbol{I}^t = \neg\boldsymbol{X}$. Next, to obtain $\boldsymbol{C}^t = \boldsymbol{I}^t$, we set $\boldsymbol{W}^{xf} = \boldsymbol{W}^{hf} = \boldsymbol{W}^{xc} = \boldsymbol{W}^{hc} = \boldsymbol{0}_{N\times N}, \boldsymbol{b}^f = -q\boldsymbol{1}_{N\times N}$ and $\boldsymbol{b}^c = q\boldsymbol{1}_{N\times N}$. This leads to the desired result:

$$\boldsymbol{F}^t = \lim_{q\to\infty}\sigma\left(-q\boldsymbol{1}_{N\times N}\right) = \boldsymbol{0}_{N\times N}, \tag{16}$$

$$\tilde{\boldsymbol{C}}^t = \lim_{q\to\infty}\tanh\left(q\boldsymbol{1}_{N\times N}\right) = \boldsymbol{1}_{N\times N},$$

$$\boldsymbol{C}^t = \boldsymbol{0}_{N\times N} \odot \boldsymbol{C}^{t-1} + \boldsymbol{I}^t \odot \boldsymbol{1}_{N\times N} = \boldsymbol{I}^t = \neg\boldsymbol{X}. \tag{17}$$

Thus, the coloring method can be implemented with a network as small as one ConvLSTM with one kernel. A network with more than one kernel and multiple stacked ConvLSTM can also solve the insideness problem for any given curve. The kernels that are not needed to implement the coloring method can be just set to 0 and the ConvLSTM that are not needed should implement the identity operation, *ie.* the output layer is equal to the input. To implement the identity operator, equation 11 can be rewritten in the following way:

$$\boldsymbol{O}^t = \lim_{q\to\infty}\sigma\left(q\boldsymbol{1}_{1\times 1} * \boldsymbol{X} - \frac{q}{2}\boldsymbol{1}_{N\times N}\right) \tag{18}$$

where $\boldsymbol{W}^{ho} = \boldsymbol{0}_{1\times 1}$ is to remove the connections with the hidden units, and q is the constant that tends to infinity. Observe that if $X_{i,j} = 1$, then $\boldsymbol{O}^t = \lim_{q\to\infty}\sigma(q/2) = 1$. If $X_{i,j} = 0$, then $\boldsymbol{O}^t = \lim_{q\to\infty}\sigma(-q/2) = 0$. Thus, the ConvLSTM implements the identity operation.

Examples of Jordan Curves of Each Dataset

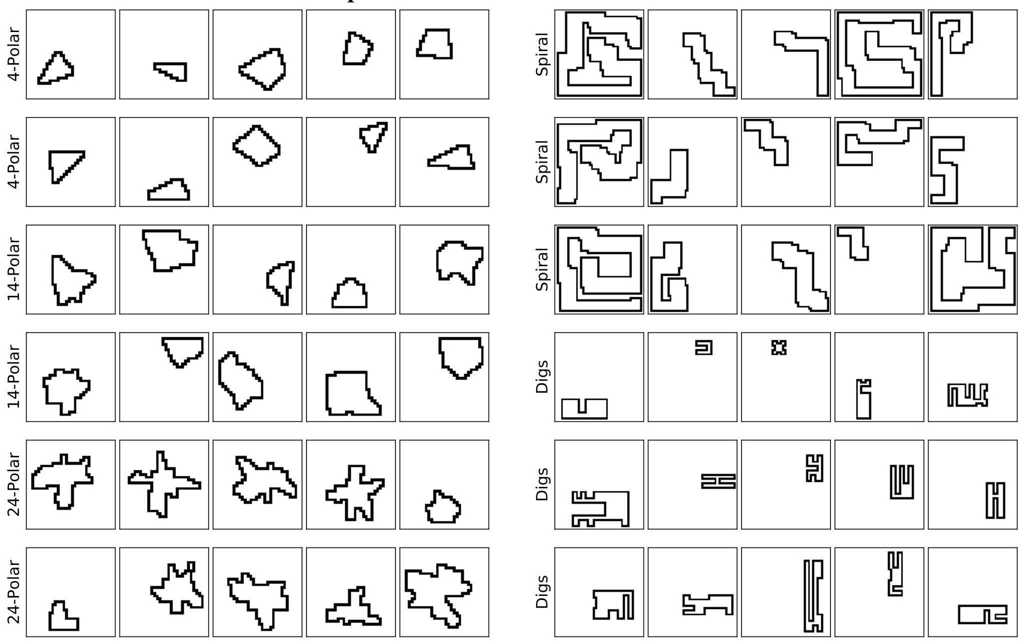


Figure E.6: *Datasets.* Images of the curves used to train and test the DNNs. Each row correspond to a different dataset.

D Coloring Routine with a Sigmoidal Convolutional RNN

There are other recurrent networks simpler than a ConvLSTM that can also implement the coloring algorithm. We introduce here a convolutional recurrent network that uses sigmoids as non-linearities. Since it is a convolutional network, for the sake of simplicity we just describe the operations done to obtain an output pixel in a step. The network has only one hidden layer, which also corresponds to the output of the network. Let $\{h_k^t\}_{k\in\mathcal{N}_{i,j}}$ be the hidden state of the output pixel indexed by i, j and its 4-neighbourhood, at step t. Let $X_{i,j}$ be the only relevant input image pixel. A necessary condition is that the outputs of the sigmoid should asymptotically be close to 0 or 1, otherwise the coloring routine would fade after many steps. It is easy to check that $h_{i,j}^{t+1} = \sigma\left(q\left(\sum_{k\in\mathcal{N}_{ij}} h_k^t - 5X_{i,j} - 1/2\right)\right)$ implements the coloring routine, where q is the factor that ensures saturation of the sigmoid.

E Dataset Generation

In Fig. E.6, we show more examples of curves in the datasets. For testing and validation sets, we only use images that are dissimilar to all images from the training set. Two images are considered dissimilar if at least 25% of the pixels of the curve are in different locations. In the following we provide a description of the algorithms to generate the curves:

- *Polar Dataset* (32×32 pixels): We use polar coordinates to generate this dataset. We randomly select the center of the figure and a random number of vertices that are connected with straight lines. These lines are constrained to follow the definition of digital Jordan curve in Sec. 2 in the main paper (and App. A in this supplementary material). The vertices are determined by their angles, which are randomly generated. The distance with respect to the center of the figure are also randomly generated to be between 3 to 14 pixels away from the center.

We generate 5 datasets with different maximum amount of vertices, namely, 4, 9, 14, 19 and 24. We refer to each of these datasets as *Polar* with a prefix with the amount of vertices.

- *Spiral Dataset* (42×42 pixels): The curves in these data set are generated from a random walk. First, a starting position is chosen uniformly at random from $[10, 20] \times [10, 20]$. Then, a segment of the spiral is built in the following way: a random direction (up, down, left, right) and a random length r from 3 to 10 are chosen so that the walk is extended by turning r pixels in the given direction.

However, such extension can only happen if adding a random thickness $t \in \{1, 2, 3, 4\}$ to both sides of this segment does not cause self intersections. These segments are added repeatedly until there is no space to add a new segment without violating the definition of a Jordan curve.

- *Digs Dataset* (42×42 pixels): We generate a rectangle of random size and then, we create "digs" of random thicknesses in the rectangle. The number of "digs" is a random number between 1 to 10. The digs are created sequentially and they are of random depth (between 1 pixel to the length of the rectangle minus 2 pixels). For each new "dig", we made sure to not cross previous digs by adjusting the depth of the "dig".

F Hyperparameters

The parameters are initialized using Xavier initialization [46]. The derived parameters we obtained in the theoretical demonstrations obtain 100% accuracy but we do not use them in this analysis as they are not learned from examples. The ground-truth consists on the insideness for each pixel in the image, as in equation 1. For all experiments, we use the cross-entropy with softmax as the loss function averaged accross pixels. Thus, the networks have two outputs per pixel (note that this does not affect the result that the networks are sufficiently complex to solve insideness, as the second output can be set to a constant threshold of 0.5). We found that the cross-entropy loss leads to better accuracy than other losses. Moreover, we found that using a weighted loss improves the accuracy of the networks. The weight, which we denote as α, multiplies the loss relative to inside, and $(1 - \alpha)$ multiplies the loss relative to outside. This α is a hyperparamter that we tune and can be equal to 0.1, 0.2 and 0.4. We try batch sizes of 32, 256 and 2048 when they fit in the GPUs' memory (12GB), and we try learning rates from 1 to 10^{-5} (dividing by 10). We train the networks for all the hyperparameters for at least 50 epochs, and until there is no more improvement of the validation set loss.

In the following we report all the tried hyperparameters for all architectures. In all cases, the convolutional layers use zero-padding.

- *Dilated Convolution DNN (Dilated):* This network was introduced in Sec. B. We use the same hyperparameters as in [7]: 3×3 kernels, a number of kernels equal to $2^l \times \{2, 4, 8\}$, where l is the number of layers and ranges between 8 to 11, with $d = 2^l$ (the first layer and the last two layers $d = 1$). The number of kernels in the layer that calculates the parity can be $\{5, 10, 20, 40, 80\}$.

- *Ray-intersection network (Ray-int.):* This is the architecture introduced in Sec. B, which uses a receptive field of $1 \times N$ instead of the dilated convolutions. The rest of the hyperparameters are as in *Dilated*.

- *Convolutional DNN (CNN):* To analyze the usefulness of the dilated convolutions, we use the *Dilated* architecture with all dilation factors $d = 1$. Also, we try adding more layers than in *Dilated*, up to 25.

- *UNet:* This is a popular architecture with skip connections and de-convolutions. We use similar hyperparameters as in [6]: starting with 64 kernels (3×3) at the first layer and doubling this number after each max-pooling; a total of 1 to 3 max-pooling layers in all the network, that are placed after sequences of 1 or 2 convolutional layers.

- *Convolutional LSTM (1-LSTM):* This is the architecture with just one ConvLSTM, introduced in Sec. 3. We use backpropagation through time by unrolling 60, 30 or 10 time steps for training (we select the best performing one). For testing, we unroll until there is no change in the output labeling. We initialize the hidden and cell states to 0 (inside) everywhere except the border of the image which is initialized to 1 (outside).

- *2-layers Convolutional LSTM (2-LSTM):* We stack one convolutional LSTM after another. The first LSTM has 64 kernels, and the hidden and cell states are initialized as in the 1-LSTM.

- *2-layers Convolutional LSTM without initialization (2-LSTM w/o init.):* this is the same as the *2-LSTM* architecture the hidden and cell states are initialized to 0 (outside).

G Regularizing Feed-Forward Networks

In Fig. 3, we have observed that *Dilated* trained on both *24-Polar* and *Spiral* datasets, obtains a test accuracy of less than 95% on these datasets while the accuracy in the training set is very close to 100%. We added weight decay in all the layers in order to regularize the network. We tried values between 10^{-5} to 1, scaling by a factor of 10. In all these experiments we have observed overfitting except for a weight decay of 1, in which the training never converged.

Also, note that the *CNN* does not have this overfitting problem. Yet, the number of layers needed is 25, which is more than the double than for *Dilated*, which is 9 layers. We added more layers to *Dilated* but the accuracy did not improve.

H Additional Figures and Visualizations

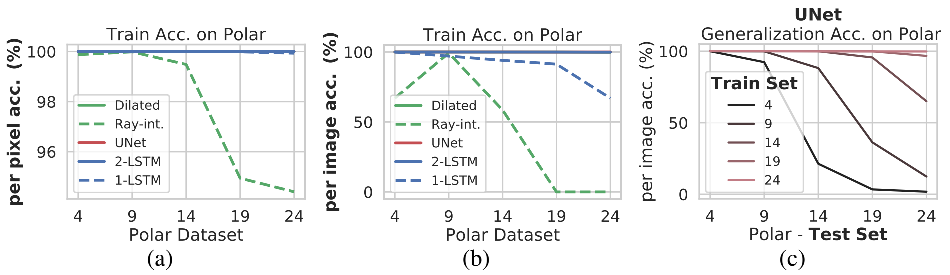


Figure H.7: *Training Accuracy in the Polar Dataset.* Intra-dataset evaluation using (a) per pixel accuracy and (b) per image accuracy on the training set, which are very similar to the test accuracy reported in Fig. 2b and c. (c) Intra-dataset evaluation of *Unet*.

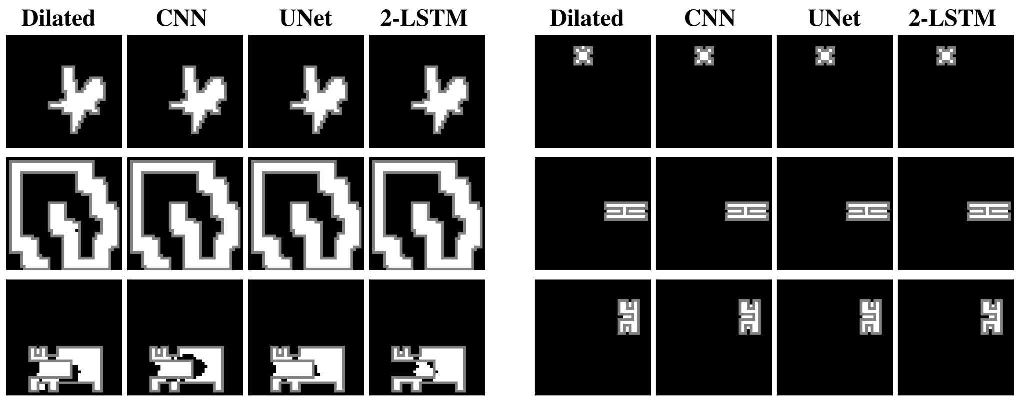


Figure H.8: *Qualitative Examples.* Networks trained in 24-Polar and Spiral dataset fail to segment in the Digs dataset.

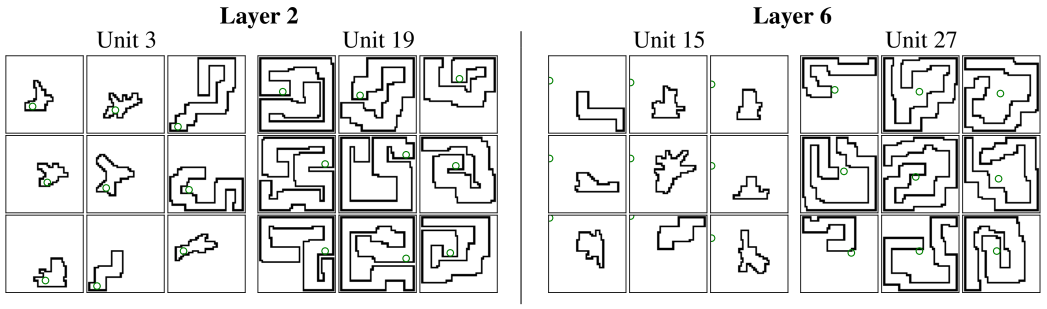


Figure H.9: *Visualization of the Units Learnt by* Dilation. Each block are the 9 images that produce the maximum activation of a units in a convolutional layer across the test set. The green dot indicates the location of the unit in the feature map. Fig. H.10 shows more examples.

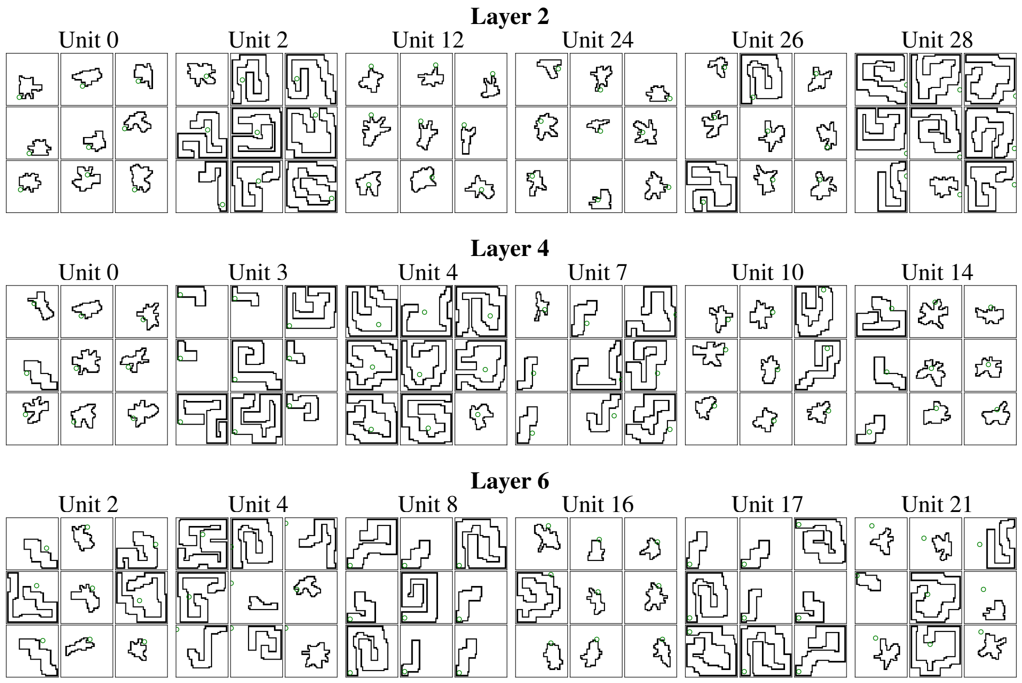


Figure H.10: *More examples of Visualization of the Units Learnt by* Dilation. The green dot indicates the location of the unit in the feature map.

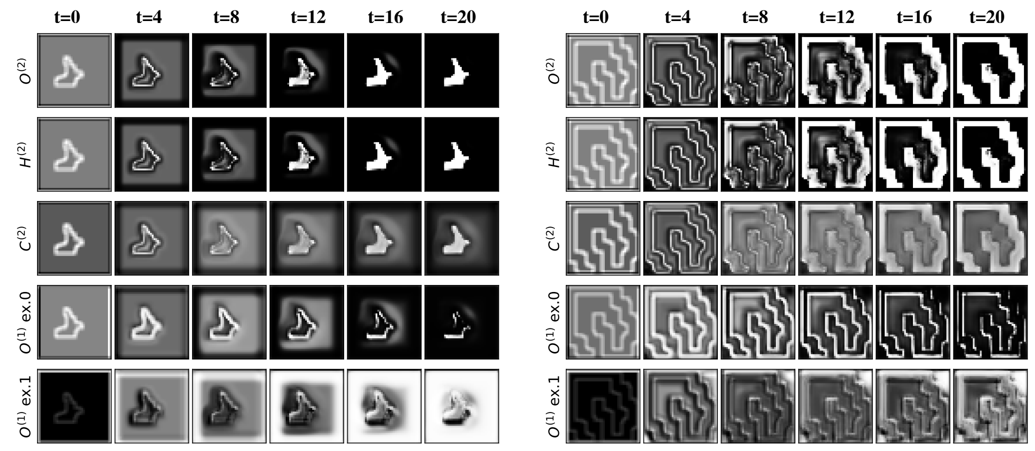


Figure H.11: *Activation Maps of the Learnt Representations by* 2-LSTM. Each row corresponds to a different layer and each colum to a different time step. For the first stacked LSTM, we show two different features maps (ex.1 and ex.0).

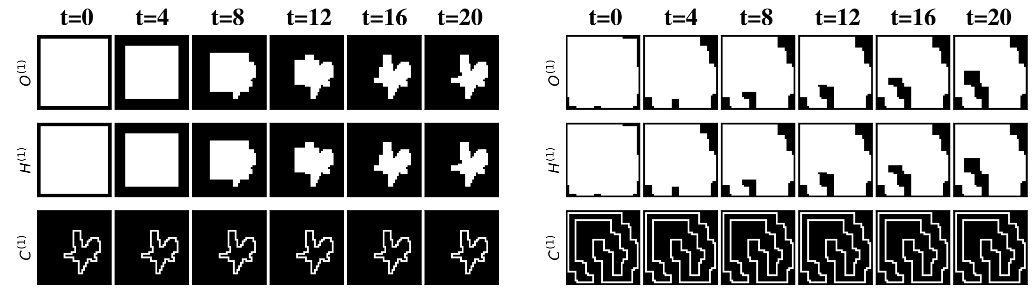
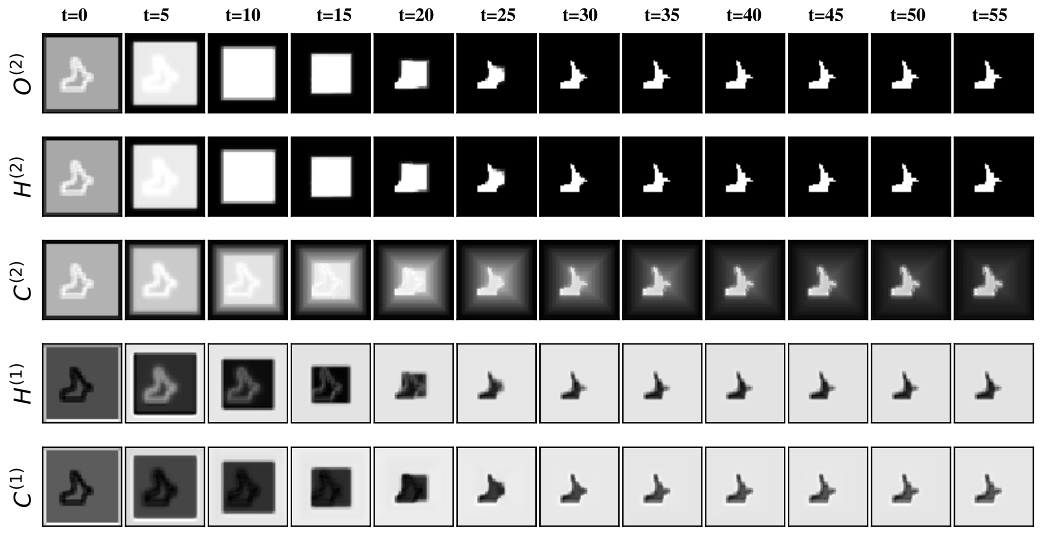


Figure H.12: *Visualization of Convolutional LSTM with the Mathematically Derived Parameters.* We can see that only the border of the image (outside) is propagated, and not the curve, as in the learnt solution.



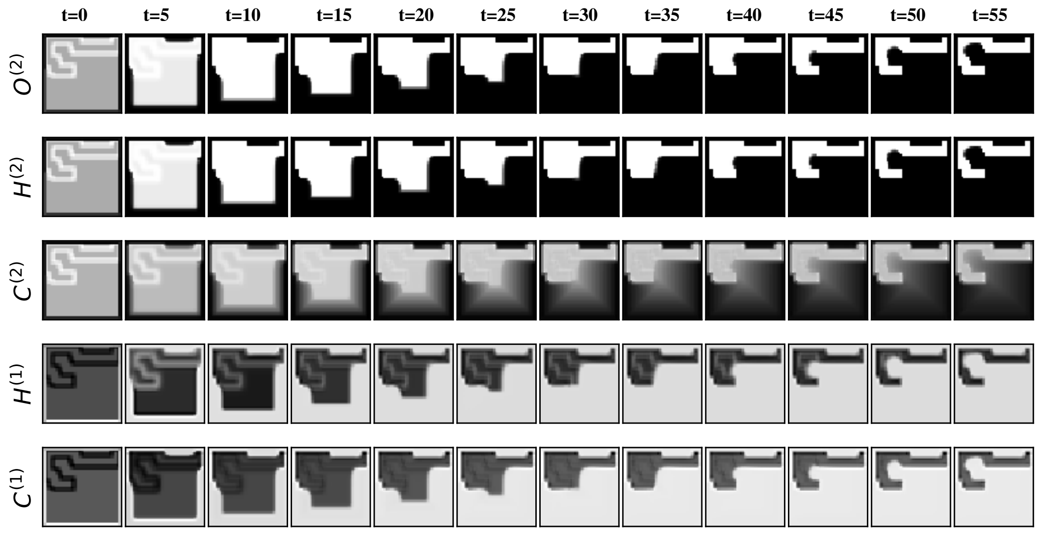


Figure H.13: *Activation Maps of the Learnt Representations by* 2-LSTM*, Trained on Small Images (*18×18 *pixels).* Each row corresponds to a different layer and each column to a different time step. Note that the network implements the coloring routine because it expands the "outside" region and blocks the expansion at the border of the curve. Interestingly, the implementation of the coloring routine is different from the one derived in the paper, which was depicted in Fig. H.12. Observe that for the solution displayed in this figure, the cell state has the same trend as the hidden state but in the derived solution the cell state is equal to $\neg \boldsymbol{X}$. This demonstrates that there are different ways of implementing the coloring routine. Also, the feature maps of the hidden state of the *RNN* (not displayed here) are very similar to the ones displayed in this figure for the second LSTM.

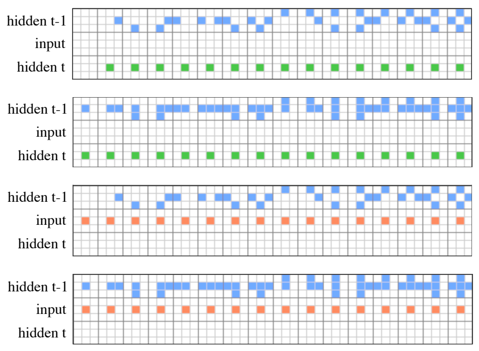


Figure I.14: *Learning the Coloring Routine*. 64 possible inputs and outputs of the training set of the per-step training of the *RNN* for the relevant inputs.

I Per-step Learning of the *Coloring Routine*

The *Coloring Routine* can be learned by enforcing to each step the ground-truth produced by the routine, rather than waiting until the last step. The inputs of a step are the image and the hidden state of the previous step. Recall that the *Coloring Routine* determines that a pixel is outside if there is at least one neighbor assigned to outside that is not at the curve border. All input cases (64) are depicted in Fig. I.14, leaving the irrelevant inputs for the *Coloring Routine* at 0. During learning, such irrelevant pixels are assigned randomly a value of 0 or 1.

We could not make any of the previously introduced LSTM networks fit a step of the coloring routine due to optimization problems. Yet, we found that the *RNN* network was able to learn with the per-step training. The *RNN* reached 0 training error about 40% of the times after randomly initializing the parameters. After training the *RNN* in one step, we unroll it and apply it to images of Jordan curves. We can see that with less than 1000 examples the *RNN* is able to generalize to any of the datasets for more than 99% of the images. This demonstrates the great potential of decomposing the learning to facilitate the emergence of the routine.

J Networks Pre-trained on Natural Images

We evaluate if training with the large-scale datasets in natural images solves the generalization issues that we reported before.

We chose two state-of-the-art networks on Instance Segmentation, DEXTR [18] and DeepLabv3+ [37], to investigate their ability in solving the insideness problem. In the following, we show that these methods fail to determine the insideness for a vast majority of curves, even after fine-tuning in the *Both* dataset (Deeplabv3+ achieved 36.58% per image accuracy in *Both* dataset and 2.18% in *Digs*.

DEXTR. Deep Exteme Cut (DEXTR) is a neural network used for interactive instance segmentation. We use the pre-trained model on PASCAL VOC 2012 [47] and show some of the qualitative results in Fig. J.15.

DeepLabv3+. This architecture extends DeepLabv3 [10] by utilizing it as an encoder network and adding a decoder network to refine segmentation boundaries. The encoder employs dilated convolution and Atrous Spatial Pyramid Pooling module for feature extraction. we use DeepLabv3+ with Xception backbone pretrained on PASCAL VOC 2012, and fine-tune its last layer with Polar and Spiral datasets for training. The ratio of input image spatial resolution to encoder output image is referred to as output stride and varies according to dilation rates. We use output strides of 8 and 16 as suggested in the paper; loss weight (α) of 0.1, 0.2 and 0.4; and initial learning rates from 0.1 to 10^{-5} (dividing by 10). We train the network on Polar and Spiral datasets until there is no improvement of the accuracy at the validations set, and we then reduce the learning rate by a ratio of 10 and stop at the next plateau of the validation set accuracy.

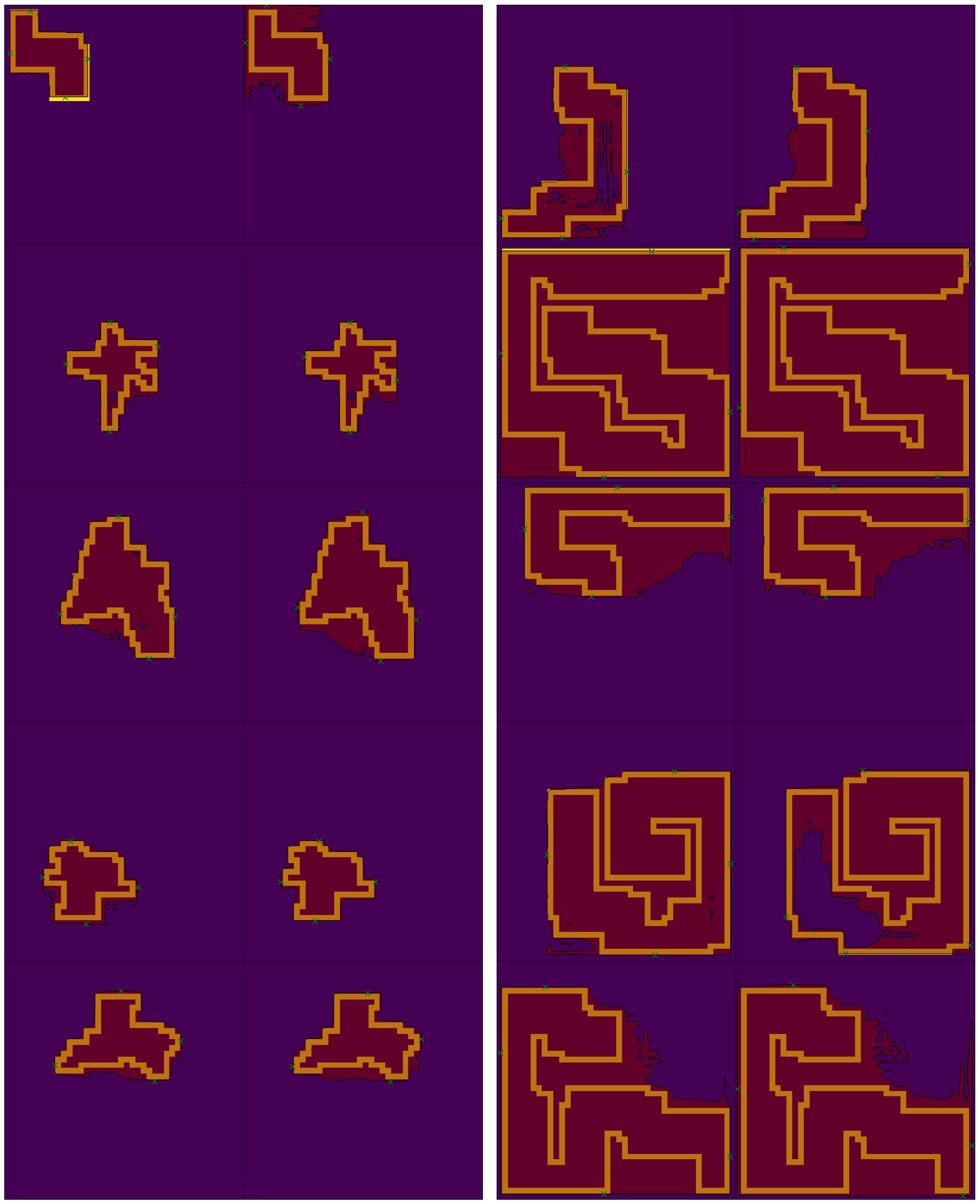


Figure J.15: *Qualitative Results with DEXTR on the Polar Dataset.* We use the publicly available pre-trained DEXTR model [18]. DEXTR uses 4 points marked by the user (indicated with crosses). We report the best found points, two examples of them per image.

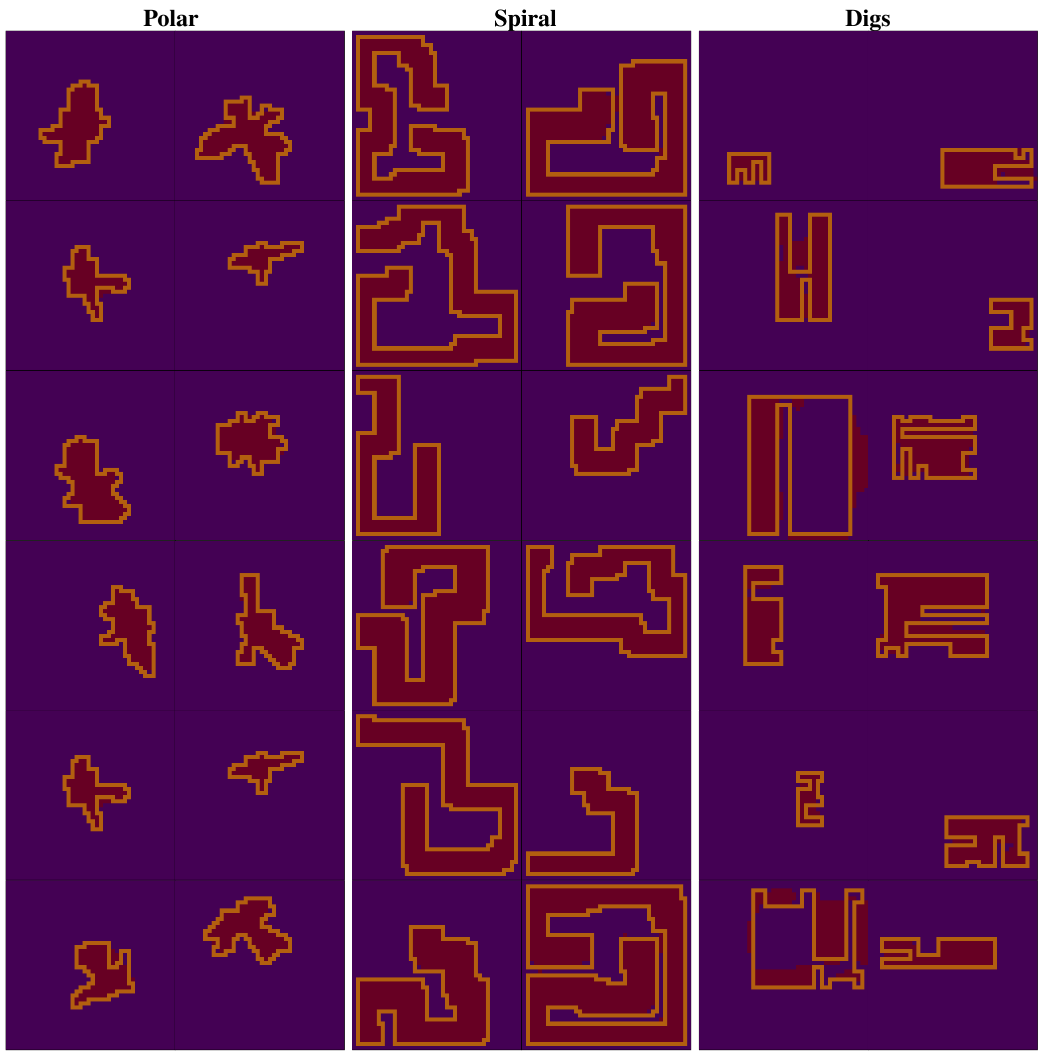


Figure J.16: *Results of DeepLabv3+ on Polar, Spiral and Digs Datasets.* The network is fine-tuned on Polar and Spiral. The results show that the network predicts well most of the pixels except in the borders. For the cross-dataset evaluations in the Digs dataset, the network is not able to generalize.