CENTER FOR
Brains
Minds+
Machines

CBMM Memo No. 95 **April 8, 2020**

Can Deep Neural Networks Do Image Segmentation by Understanding Insideness?

Kimberly M Villalobos, Jamell Dozier, Vilim Štih, Andrew Francl, Frederico Azevedo Tomaso Poggio, Tomotake Sasaki, Xavier Boix

Center for Brains, Minds, and Machines

Abstract

THIS MEMO IS REPLACED BY CBMM MEMO 105
https://cbmm.mit.edu/sites/default/files/publications/CBMM-Memo-105.pdf

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

CAN DEEP NEURAL NETWORKS DO IMAGE SEGMENTATION BY UNDERSTANDING INSIDENESS?

Kimberly M Villalobos, Jamell Dozier, Vilim Štih, Andrew Francl, Frederico Azevedo
Tomaso Poggio, Tomotake Sasaki, Xavier Boix

Center for Brains, Minds, and Machines

ABSTRACT

A key component of visual cognition is the understanding of spatial relationships among objects. Albeit effortless to our visual system, state-of-the-art artificial neural networks struggle to distinguish basic spatial relationships among elements in an image. As shown here, deep neural networks (DNNs) trained with hundreds of thousands of labeled examples cannot accurately distinguish whether pixels lie inside or outside 2D shapes, a problem that seems much simpler than image segmentation. In this paper, we sought to analyze the capability of ANN to solve such inside/outside problem using an analytical approach. We demonstrate that it is a mathematically tractable problem and that two previously proposed algorithms, namely the Ray-Intersection Method and the Coloring Method, achieve perfect accuracy when implemented in the form of DNNs.

1 INTRODUCTION

Visual cognition is the ability of analyzing shape properties and spatial relations of components of an image Ullman (1983). A central goal of computer vision has been to implement it algorithmically. While much emphasis has been put in the development of algorithms dealing with the analysis of shape properties, *i.e.* object identification and categorization LeCun et al. (2015), less focus has been placed in the implementation of algorithms that efficiently capture spatial relationships of elements in an image. In a natural environment, understanding the spatial configurations of objects is fundamental for tasks such as navigation, manipulation of objects, action planning and spatial reasoning, etc.

One of the basic abilities of spatial reasoning is capturing the concept of insideness, *i.e.* being able to determine whether a given dot lies inside or outside another shape. Although it seems to be a trivial problem, state-of-the-art deep neural networks (DNNs) struggle to determine such relationship Minsky & Papert (1988).

In this paper, we analyze the ability of current deep neural networks to solve the insideness problem well. The mathematical tractability allows us to analytically construct deep neural networks that solve at perfection the insideness problem for any arbitrary shape. These deep neural networks borrow the algorithmic ideas of the Ray-Intersection Method and the Coloring method to deep neural networks Ullman (1983). We show that the crossing and coloring algorithm have an equivalent artificial neural network that implements them using a number of neurons linear in the size of the image. By employing the optimal architectures and a set of weights analytically determined, DNNs are able to discern inside from outside with perfect accuracy. Further, these DNNs are realizable in practice, as the number of neurons in the DNNs grows linearly with the image size and complexity of the shape. Although these results are encouraging, it remains to be seen if these DNNs are learnable or not.

2 INSIDENESS

In this Section, we introduce the problem of "insideness", which aims at capturing the most basic form of segmentation. We use synthetic stimuli that solely contains a closed curve, which discard

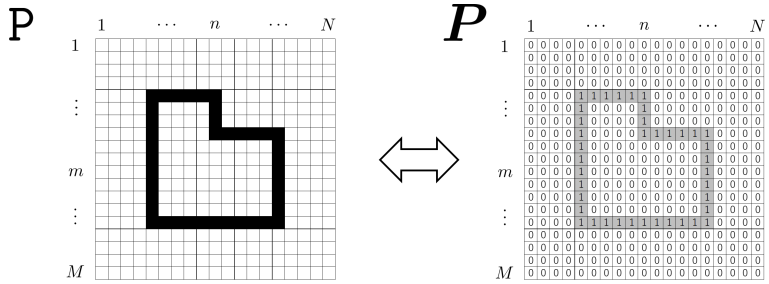


Figure 1: Picture and its binary matrix representation.

issues related to object semantics, visual textures, hierarchy of segments, etc. Although these factors are commonly tackled by segmentation algorithms, they can be discarded without affecting the core problem of segmentation: distinguishing among parts of an image.

Insideness consists on finding which pixels are in the inside and which ones in the outside of a closed curve. We assume for simplicity's sake and without loss of generality of the conclusions of this paper, that there is only one closed curve in the image and it is a Jordan curve, *ie.*a closed curve without self-crossing or self-touching (see Appendix A for the exact definition of Jordan curves). Let $\mathbf{X} \in \{0,1\}^{M \times N}$ be a binary image of size $M \times N$ pixels that contains one and only one Jordan curve. We use the sub-indices m, n to denote the pixel at the coordinates (m, n), *e.g.*$X_{m,n}$. The Jordan curve in $\mathbf{X}$ correspond to the pixels equal to 1, $\{X_{m,n} = 1\}$, otherwise the pixels are 0. In Figure 1, we show examples of images for insideness.

The pixels in $\mathbf{X}$ with value $X_{m,n} = 0$ can be classified into two categories: the inside region and the outside region of the Jordan curve. Let $\mathbf{S}(\mathbf{X}) \in \{0,1\}^{M \times N}$ be an image that represents the segmentation of $\mathbf{X}$. The segmentation of each pixel, $S(\mathbf{X})_{m,n}$, can be defined as follows:

$$S(\mathbf{X})_{m,n} = \begin{cases} 0 & \text{if } X_{m,n} = 0 \text{ and } X_{m,n} \text{ is in the inside} \\ 1 & \text{if } X_{m,n} = 0 \text{ and } X_{m,n} \text{ is in the outside} \\ 0 \text{ or } 1 & \text{if } X_{m,n} = 1 \end{cases}, \tag{1}$$

where the pixels of the Jordan curve, $\{X_{m,n} = 1\}$, can be segmented either as inside or outside and are not evaluated. Note that unlike object recognition, insideness is rigorously and uniquely determined by the input image itself.

3 The Ray-Intersection Neural Network

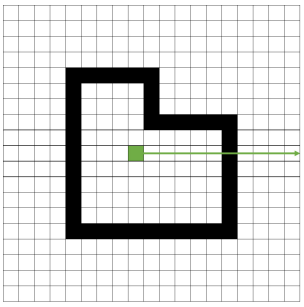
The ray-intersection algorithm is also known as the crossings test or the even-odd test (Haines, 1994; Fulton, 1995; Klette & Rosenfeld, 2004), and it has been used to distinguish inside and outside of a curve (Ullman, 1983; 1984; 1996). In the following subsections, we introduce the ray algorithm adapted to our nomenclature and then, its implementation with an DNN.

3.1 The Ray Algorithm

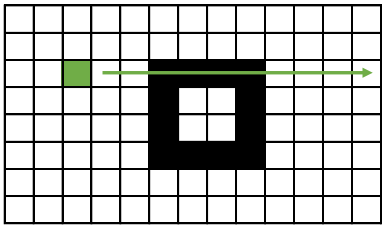
Given an image $\mathbf{X} \in \{0,1\}^{M \times N}$ containing one and only one digital Jordan curve $\mathtt{F}_{\mathbf{X}}$, a horizontal or vertical ray through the image alternates between inside region and outside region every time it crosses the curve $\mathtt{F}_{\boldsymbol{X}}$. Therefore, to find if a pixel with value 0 lies within the inside region of $\boldsymbol{X}$, it is enough to count the number of times that any ray that starts from the pixel crosses $\mathtt{F}_{\boldsymbol{X}}$ (the crossing number) and check its parity. If the parity of the crossing number is odd then the pixel is inside, otherwise it is outside (see Fig. 2(a)). While the definition of a cross is intuitively simple, we can see in Fig. 2(b) an example of a ray that touches the curve, but the type of region does not change after this intersection. We then proceed to formally characterize a ray as well as what we mean by crossing a curve.

Definition 1. Given a pixel $X_{m,n}$, we define the row vector associated to a ray pointing to the right from this pixel as

$$\vec{X}_{m,n} = [X_{m,n+1}, X_{m,n+2}, \ldots, X_{m,N}, 0, \ldots, 0] \in \{0,1\}^{1 \times N}. \tag{2}$$



(a) Ray Algorithm: the crossing number is 1 and we can tell the pixel is inside.



(b) Example of a case in which type of region does not change after the intersection.

Figure 2: Valid and Invalid intersections

While the direction of the rays does not change the crossing number, we will use horizontal rays throughout this paper for consistency. Let's now consider all possible cases for how the ray $\vec{X}_{m,n}$ could cross $\mathtt{F}_{\boldsymbol{X}}$; i.e, we analyze all possible segments of consecutive entries of $\vec{X}_{m,n}$ that have all value 1. Suppose then that we have s consecutive entries $X_{m,n'+1} = X_{m,n'+2} = \ldots = X_{m,n'+s} = 1$ with $X_{m,n'} = X_{m,n'+s+1} = 0$. Since each pixel in $\mathtt{F}_{\boldsymbol{X}}$ has exactly 2 adjacent entries that are also in $\mathtt{F}_{\boldsymbol{X}}$, we have only the following 5 cases depicted in Fig. 3:

1. $s > 1$, $X_{m+1,n'+1} = X_{m-1,n'+s} = 1$ and $X_{m-1,n'+1} = X_{m+1,n'+s} = 0$
2. $s = 1$, $X_{m-1,n'+1} = X_{m+1,n'+1} = 1$
3. $s > 1$, $X_{m+1,n'+1} = X_{m-1,n'+s} = 0$ and $X_{m-1,n'+1} = X_{m+1,n'+s} = 1$
4. $s > 1$, $X_{m+1,n'+1} = X_{m+1,n'+s} = 0$ and $X_{m-1,n'+1} = X_{m-1,n'+s} = 1$
5. $s > 1$, $X_{m+1,n'+1} = X_{m+1,n'+s} = 1$ and $X_{m-1,n'+1} = X_{m-1,n'+s} = 0$

We then notice that each time one of the first 3 cases occurs, the region indeed switches from inside to outside or from outside to inside. On the other hand, the last two represent invalid intersections (like in Fig. 2(b)) that should not be counted in the calculation of the crossing number, or at least should not affect its parity. We say that a ray crosses $\mathtt{F}_{\boldsymbol{X}}$ if we are in one of the first three valid cases. This implies that a pixel $X_{m,n}$ with value 0 is in the inside region if and only if the number of times that $\vec{X}_{m,n}$ crosses $\mathtt{F}_{\boldsymbol{X}}$ is odd. The following Lemma tells us how to find such a parity.

Lemma 1. Consider a rightward horizontal ray from a pixel $X_{m,n}$ with value 0. Then,

$$\vec{X}_{m,n} \cdot \vec{X}_{m+1,n} \equiv \text{crossing number of } \vec{X}_{m,n} \pmod 2, \tag{3}$$

where $\cdot$ denotes the standard inner product.

Proof. If in addition to the horizontal ray $\vec{X}_{m,n}$ from $X_{m,n}$ we consider also the horizontal ray $\vec{X}_{m+1,n}$ below it, then we get the following invariant: for the first three crossing cases it holds

$$\sum_{i=1}^{s} X_{m,n'+i} \cdot X_{m+1,n'+i} \equiv 1 \pmod 2 \tag{4}$$

whereas for the last two cases it holds

$$\sum_{i=1}^{s} X_{m,n'+i} \cdot X_{m+1,n'+i} \equiv 0 \pmod 2 \tag{5}$$

and therefore, $\vec{X}_{m,n} \cdot \vec{X}_{m+1,n}$ has the same parity as the number of times that $\vec{X}_{m,n}$ crosses $\mathtt{F}_{\boldsymbol{X}}$, as desired. □

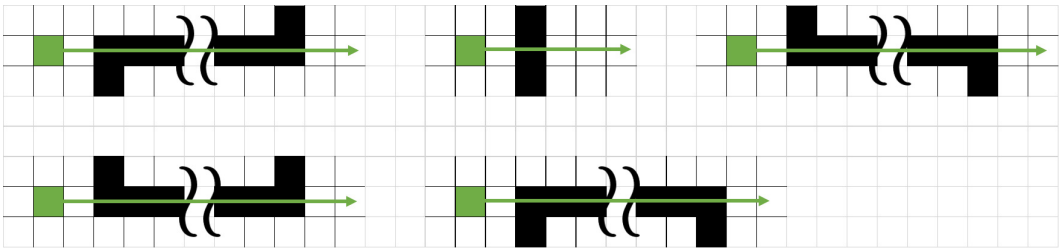


Figure 3: Depiction of the the 5 cases in which a ray can intersect the figure.

Since the parity of crossing number of $\vec{X}_{m,n}$ is equal to $S(\boldsymbol{X})_{m,n}$, the distilling-segmentation problem can be solved by an algorithm that finds the parity of $\vec{X}_{m,n} \cdot \vec{X}_{m+1,n}$ for all pixels $X_{m,n}$ and outputs 1 for the pixels for which that quantity is odd, and 0 for those where the inner product is even. We then are ready to present a feed-forward neural network that implements this algorithm.

3.2 Ray Network

We are now ready to describe a neural network that implements the Ray Method. We define a 3-layer feedforward neural network (*Ray Network*) that for an input image $\boldsymbol{X}$ and a set of parameters $\boldsymbol{\Theta}$, computes the function $\hat{\boldsymbol{S}}_R(\boldsymbol{X}; \boldsymbol{\Theta})$, and we find optimal weights $\boldsymbol{\Theta}_R$ such that $\boldsymbol{S}(\boldsymbol{X}) = \hat{\boldsymbol{S}}_R(\boldsymbol{X}; \boldsymbol{\Theta}_R)$ perfectly estimates the desired answer satisfying (1). Roughly speaking, the 1st layer will preprocess the input image so as to find ray intersections that are also crosses, and the second and output layer will calculate the parity of the number of crossings for the ray corresponding to each pixel.

The first hidden layer of this network, $\boldsymbol{H}^{(1)} \in \{0,1\}^{M \times N}$, is defined by

$$\boldsymbol{H}^{(1)} \coloneqq \left[\boldsymbol{w}^{(1)} * \boldsymbol{X} + \boldsymbol{B}^{(1)}\right]_+, \tag{6}$$

where the symbol $*$ denotes the so-called convolution, $\boldsymbol{w}^{(1)} \in \mathbb{R}^2$ and the function $[\quad]_+$ represents element-wise application of the $ReLU$ function to a matrix or a vector.

Let $\boldsymbol{1}_I \in \mathbb{R}^I$ and $\boldsymbol{1}_{I \times J} \in \mathbb{R}^{I \times J}$ denote the vector or matrix containing only 1s. If we set $\boldsymbol{w}_R^{(1)} = \boldsymbol{1}_2$ and $\boldsymbol{B}_R^{(1)} = -\boldsymbol{1}_{M \times N}$, i.e, if the receptive field of the filter is simply the pixel and the pixel below it, then we have that $H_{m,n}^{(1)} = 1$ if and only if $X_{m,n} = X_{m+1,n} = 1$, implying that

$$\boldsymbol{1}_{1 \times N} \cdot \vec{H}_{m,n}^{(1)} = \vec{P}_{m,n} \cdot \vec{P}_{m+1,n}. \tag{7}$$

Therefore, by Lemma 1 the network just needs to compute the parity of $\boldsymbol{1}_{1 \times N} \cdot \vec{H}_{m,n}^{(1)}$ in the following layers.

A feed-forward NN with one hidden layer and input $\boldsymbol{x} \in \{0,1\}^{1 \times N}$ is presented in (Shalev-Shwartz et al., 2017) to find the parity of $\boldsymbol{1}_{1 \times N} \cdot \boldsymbol{x}$. The first layer is a fully connected layer with $3C$ neurons, where C is an upper bound for $\boldsymbol{1}_{1 \times N} \cdot \boldsymbol{x}$, and the output layer is a fully connected layer with a single neuron. Shalev-Schwartz et al provide a set of optimal parameters for which this neural network outputs 1 if $\boldsymbol{1}_{1 \times N} \cdot \boldsymbol{x}$ is odd and 0 if it is even. In what follows we proceed to extend such a network to a higher order case so that we can compute the parity of $\boldsymbol{1}_{1 \times N} \cdot \vec{H}_{m,n}^{(1)}$ for all m, n.

Let $C(\boldsymbol{X})$ be the maximum number of times that a horizontal or vertical ray can cross the curve $\mathtt{F}_{\boldsymbol{X}}$ and let $C' = 3C(\boldsymbol{X})$. Then, we know that $\boldsymbol{1}_{1 \times N} \cdot \vec{H}_{m,n}^{(1)} \leq C(\boldsymbol{X})$. We then define the second layer $\mathbf{H}^{(2)} \in \mathbb{R}^{M \times N \times C'}$ by the equation

$$\boldsymbol{H}_{:,:,r}^{(2)} \coloneqq \left[\boldsymbol{w}^{(2)} * \boldsymbol{H}^{(1)} + \boldsymbol{B}_r^{(2)}\right]_+, \quad \text{for all } r = 1, 2, \ldots, C', \tag{8}$$

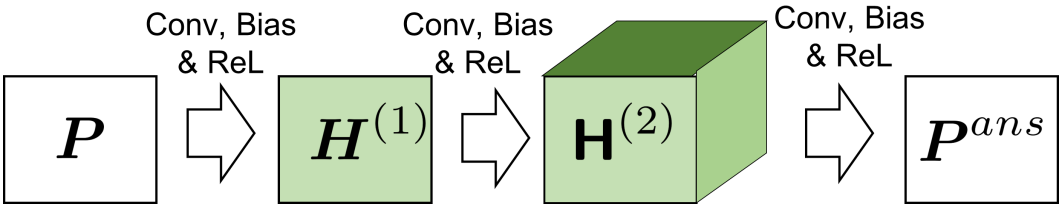


Figure 4: Depiction of the Ray Network.

where $\boldsymbol{A}_{:,:,k}$ denotes the k^{th} "matrix-slice" of $\mathbf{A}$, $\boldsymbol{w}^{(2)} \in \mathbb{R}^{1\times N}$ and $\boldsymbol{B}_r^{(2)} \in \mathbb{R}^{M\times N}$. And we lastly define the output layer as

$$\boldsymbol{Y} := \left[\boldsymbol{w}^{(3)} * \mathbf{H}^{(2)} + \boldsymbol{B}^{(3)}\right]_+, \tag{9}$$

where $\boldsymbol{w}^{(3)} \in \mathbb{R}^{1\times 1\times C'}$ and $\boldsymbol{B}^{(3)} \in \mathbb{R}^{M\times N}$. (See Fig. 4 for the complete structure of the Ray Network).

By extending the optimal weights from the network presented by Shalev-Schwartz et al, we can find optimal parameters $\boldsymbol{w}_R^{(2)}, (\boldsymbol{B}_R)_r^{(2)}, \boldsymbol{w}_R^{(3)}$ and $\boldsymbol{B}_R^{(3)}$ for our network such that $Y_{m,n}$ outputs the parity of $\boldsymbol{1}_{1\times N} \cdot \vec{H}_{m,n}^{(1)}$. (The exact parameter values can be found in Appendix B). Therefore, if we define $\boldsymbol{\Theta}_R = (\boldsymbol{w}_R^{(1)}, \boldsymbol{B}_R^{(1)}, \boldsymbol{w}_R^{(2)}, \{(\boldsymbol{B}_R)_r^{(2)}\}_{r=1}^{C'}, \boldsymbol{w}_R^{(3)}, \boldsymbol{B}_R^{(3)})$ and apply both equation 7 and Lemma 1, we obtain that with this weights $Y_{m,n}$ outputs $S_{n,m}$.

We can then summarize our findings from these section with the following theorem:

Theorem 1. There exist a feed forward neural network $\hat{\boldsymbol{S}}_R(\boldsymbol{X};\boldsymbol{\Theta}_R)$ with $O(C\times M\times N)$ neurons, and optimal parameters $\boldsymbol{\Theta}_R$ such that

$$\hat{\boldsymbol{S}}_R(\boldsymbol{X};\boldsymbol{\Theta}_R) = \boldsymbol{S}(\boldsymbol{X}).$$

4 The Coloring Neural Network

pdf

4.1 Coloring Algorithm

The *Coloring Algorithm* (Ullman, 1983; 1984; 1996) is an application of the floodfill algorithm (Torbert, 2016), and is another computational principle we can use to distinguish inside and outside. The idea of this method is that if we know the region to which a pixel $X_{n,m}$ with value 0 belongs, we can identify the region to which its neighbors belong (this is analogous to using the same color for points that belong to the same region when coloring an image). Since, by definition of our input images, the pixels in the border of $\boldsymbol{X}$ are always in the outside region, we can identify all the other pixels in the outside region recursively starting from the borders of the image and until this process reaches the curve $\mathtt{F}_{\boldsymbol{X}}$ (Fig. 5). When this recursion ends, we know that the pixels with 0 value which have not been identified to be in the outside region must be in the inside one. We can then formalize our version of the coloring algorithm by treating 1 and 0 as Boolean values $true$ and $false$ respectively, as follows:

Step 1 Initialize a memory $\boldsymbol{H} \in \mathbb{R}^{M\times N}$ with value 1 in the borders and 0 everywhere else. Then, repeat $N \cdot M$ times:

Step 2 For each pixel $X_{m,n}$:

(2a) Set $p = [H_{m,n}$ and its neighbors have all value 0].

(2b) Set $q = \neg p$.

(2c) Update $H_{m,n} = [q$ AND $X_{m,n} = 0]$.

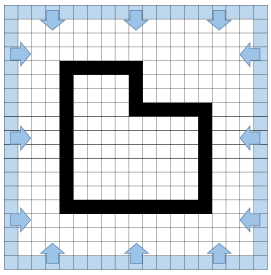


Figure 5: Coloring Method.

Intuitively, out memory $\boldsymbol{H}$ stores a value of 1 for all pixels that we know for sure are in the outside region, and 0 for all the other pixels. In Step 1 we initialize the pixels in the border of the image as outside pixels. In steps $(2a)$ and $(2b)$ we check if the pixel or its neighbors were already determined to be outside, and in step $(2c)$ we identify a pixel to be outside if the neighbors are outside and the pixel does not belong to the curve. The correctness of this algorithm follows from the fact that if there is a path of t pixels starting at some border pixel, moving vertically or horizontally through neighboring pixels and ending at some outside pixel $X_{n,m}$, then after t iterations of the loop in step 2, pixel $X_{n,m}$ is correctly classified as an outside pixel. Since the longest such path has at most $N \cdot M$ pixels, we know that when the algorithm finishes all outside pixels were found. In addition, since pixels belonging to the curve are never set to be outside, neither are the pixels inside the curve and therefore we know all unclassified pixels with value 0 must be inside.

4.2 Coloring Network

We now define a recurrent neural network (*Coloring Network*) that for an input image $\boldsymbol{X}$ and a set of parameters $\boldsymbol{\Theta}$, computes the function $\hat{\boldsymbol{S}}_C(\boldsymbol{X};\boldsymbol{\Theta})$, and once again we find optimal parameters $\boldsymbol{\Theta}_C$ such that $\boldsymbol{S}(\boldsymbol{X}) = \hat{\boldsymbol{S}}_C(\boldsymbol{X};\boldsymbol{\Theta}_C)$. At the the t^{th} iteration, this network updates a memory $\boldsymbol{H}[t] \in \mathbb{R}^{M\times N}$ that stores a value of 1 for all pixels that were found to be in the outside region and a value of 0 for all other pixels. We then implement the steps of the coloring algorithm by interpreting 1 as the Boolean value for *True* and 0 as the Boolean value *False*. As we show in Appendix C), there exist simple feed forward neural networks $\boldsymbol{F}^{\mathrm{N}}$ and $\boldsymbol{F}^{\mathrm{A}}$ and optimal parameters $\boldsymbol{\Theta}_{\mathrm{N}}, \boldsymbol{\Theta}_{\mathrm{A}}$ such that

$$\boldsymbol{NOT}(\boldsymbol{Y}) \coloneqq \boldsymbol{F}^{\mathrm{N}}(\boldsymbol{Y}, \boldsymbol{\Theta}_{\mathrm{N}})$$

implements the element-wise Boolean functions NOT for the input matrix $\boldsymbol{Z}$, and

$$\boldsymbol{AND}(\boldsymbol{Y}, \boldsymbol{Z}) \coloneqq \boldsymbol{F}^{\mathrm{A}}(\boldsymbol{Y}, \boldsymbol{Z}, \boldsymbol{\Theta}_{\mathrm{A}})$$

implements the element-wise Boolean function AND for input matrices $\boldsymbol{Y}$ and $\boldsymbol{Z}$. We now proceed to implement the Coloring Algorithm by using these neural sub-networks as building blocks.

Step 1 (Initialize a memory $\boldsymbol{H} \in \mathbb{R}^{M\times N}$ with value 1 in the borders and 0 everywhere else).
We manually initialize a memory $\boldsymbol{H}[0] \in \mathbb{R}^{M\times N}$ that has value 1 for all pixels in the border of the image and value 0 in all other pixels.

Step $(2a)$ (Set $p = [H_{m,n}$ and its neighbors have all value 0]).
For this, we define the hidden layer $\boldsymbol{P}[t] \in \mathbb{R}^{M\times N}$ by

$$\boldsymbol{P}[t] = [\boldsymbol{W} * \boldsymbol{H}[t-1] + \boldsymbol{B}]_+ ,$$

where $\boldsymbol{W} \in \mathbb{R}^{3\times 3}$ and $\boldsymbol{B} \in \mathbb{R}^{M\times N}$. Notice that the receptive field of the filter $\boldsymbol{W}$ covers all neighbors of the pixel, and therefore if we set $\boldsymbol{W}_C = -\boldsymbol{1}_{3\times 3}$ and $\boldsymbol{B}_C = \boldsymbol{1}_{M\times N}$, we have that

$$P[t]_{m,n} = 1 \iff H[t-1]_{m,n} \text{ and its neighbors have all value } 0.$$

Step $(2b)$ (Set $q = \neg p$).
The building block for this step is the neural network $\boldsymbol{F}^{\mathrm{N}}$, which takes $\boldsymbol{P}[t]$ as its input and outputs

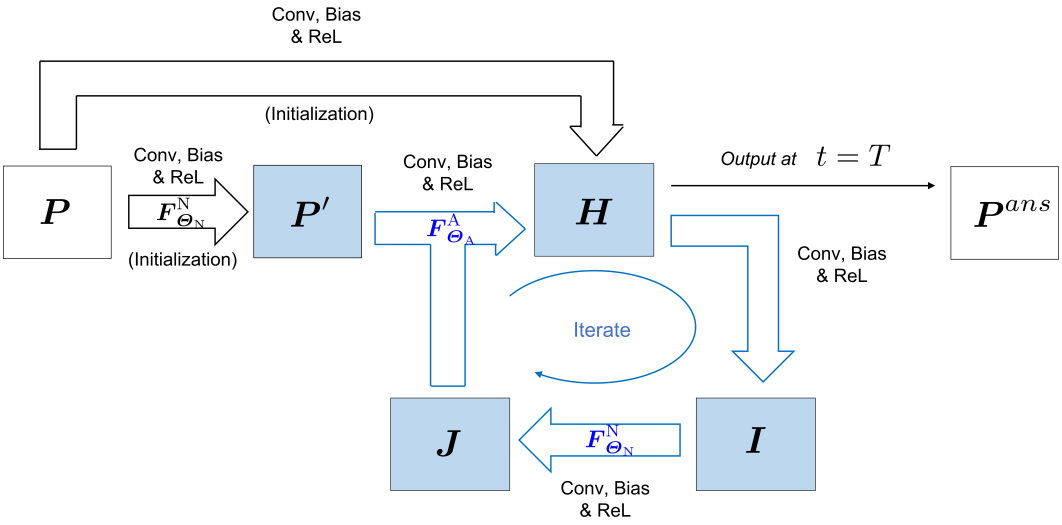


Figure 6: Depiction of the Coloring Network.

$\boldsymbol{Q}[t] \in \mathbb{R}^{M \times N}$. Therefore, if we use the optimal parameters $\boldsymbol{\Theta}_{\mathrm{N}}$, we obtain that the output of this step is given by

$$\boldsymbol{Q}[t] \coloneqq \boldsymbol{NOT}(\boldsymbol{P}[t]),$$

where

$$Q[t]_{m,n} = 1 \iff P[t]_{m,n} = 0,$$

as desired.

Step (2c) (Update $H_{m,n} = [q \text{ AND } X_{m,n} = 0]$).
This block, is the composition of the networks $\boldsymbol{F}^{\mathrm{N}}$ and $\boldsymbol{F}^{\mathrm{A}}$, that with optimal parameters $\boldsymbol{\Theta}_{\mathrm{N}}$ and $\boldsymbol{\Theta}_{\mathrm{A}}$ yield

$$\boldsymbol{H}[t+1] \coloneqq \boldsymbol{AND}(\boldsymbol{Q}[t], \boldsymbol{NOT}(\boldsymbol{X})),$$

with

$$H[t+1]_{m,n} = 1 \iff Q[t]_{m,n} = 1 \text{ and } X_{m,n} = 0.$$

We repeat the loop $M \times N$ times so that the output of our network is given by $\hat{\boldsymbol{S}}_C(\boldsymbol{X}; \boldsymbol{\Theta}_C) = \boldsymbol{H}[M \times N]$. Therefore, defining $\boldsymbol{\Theta}_C = (\boldsymbol{W}_C, \boldsymbol{B}_C, \boldsymbol{\Theta}_{\mathrm{N}}, \boldsymbol{\Theta}_{\mathrm{A}})$ we know that this network successfully implements each step of the coloring algorithm. (See Fig. 6 for the structure of the Coloring Network.) We can summarize this section with the following theorem

Theorem 2. There exists a recurrent neural network $\hat{\boldsymbol{S}}_C(\boldsymbol{X}; \boldsymbol{\Theta})$ with $O(M \times N)$ neurons, and optimal parameters $\boldsymbol{\Theta}_C$ such that

$$\hat{\boldsymbol{S}}_C(\boldsymbol{X}; \boldsymbol{\Theta}_C) = \boldsymbol{S}(\boldsymbol{X}).$$

5 Conclusion

We have shown that current DNNs can solve at perfection the insideness problem. These DNNs are easily implementable in practice, as the number of neurons scales linearly with the image size and complexity of the shapes. It remains to be seen if these networks are able to learn the optimal weights when trained with examples.

References

William Fulton. *Algebraic Topology: A First Course*. Springer, 1 edition, 1995.

Eric Haines. Point in polygon strategies. In Paul Heckbert (ed.), *Graphics Gems IV*, pp. 24–46. Academic Press, 1994.

Reinhard Klette and Azriel Rosenfeld. *Digital geometry: Geometric methods for digital picture analysis*. Elsevier, 2004.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

Marvin L. Minsky and Seymour A. Papert. *Perceptrons: an introduction to computational geometry*. MIT Press, 1st edition, 1969.

Marvin L. Minsky and Seymour A. Papert. *Perceptrons: an introduction to computational geometry*. MIT Press, expanded edition, 1988.

Shai Shalev-Shwartz, Ohad Shamir, and Shaked Shammah. Failures of gradient-based deep learning. *arXiv preprint arXiv:1703.07950*, 2017.

Shane Torbert. *Applied Computer Science*. Springer, 2nd edition, 2016.

Shimon Ullman. Visual routines. Technical Report A.I. Memo No.723, Massachusetts Institute of Technology Artificial Intelligence Laboratory, Cambridge, June 1983.

Shimon Ullman. Visual routines. *Cognition*, 18:97–159, 1984.

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

A Formal Definitions About Inside/Outside Problem

Here we give formal definitions of basic notions used in this paper. We referred to the related definitions in the field of digital geometry (Klette & Rosenfeld, 2004) and in the book "Perceptrons" (Minsky & Papert, 1969; 1988), but sometimes simplified or modified them for making the definitions most suitable for our purposes.

Definition 2. When considering the matrix representation $\boldsymbol{X} \in \{0,1\}^{M\times N}$ of a black and white image, we slightly abuse notation and refer to $\boldsymbol{X}$ as the image and to $X_{n,m}$ as the pixel in position (n,m), where we assume $(1,1)$ is the upper left corner of the image.

Definition 3 (Border of an image)**.** We refer to the border of an image $\boldsymbol{X}$ as the set of pixels $X_{n,m}$ such that $m=1, m=M, n=1$ or $n=N$.

Definition 4 (digital Jordan curve)**.** Let $\boldsymbol{X} \in \{0,1\}^{M\times N}$ for some $N, M \geq 3$, and let $c = (\mathtt{s}_0, \mathtt{s}_1, \ldots, \mathtt{s}_L)$ be a sequence of pixels in $\boldsymbol{X}$ such that they all have value 1. We call c a digital Jordan curve in $\boldsymbol{X}$ if the following five conditions are satisfied:

1. $\mathtt{s}_0 = \mathtt{s}_L$.
2. For any $i \in \{0,\ldots,L-1\}$, pixels $\mathtt{s}_i$ and $\mathtt{s}_{i+1}$ have a common edge.
3. It holds $\mathtt{s}_i \neq \mathtt{s}_j$ for all $i \neq j$.
4. For each $i \in \{0,\ldots,L-1\}$, $\mathtt{s}_i$, there exist exactly two other pixels in $\{\mathtt{s}_0, \mathtt{s}_1, \ldots, \mathtt{s}_{L-1}\}$ that have a common edge with s_i.

Note that conditions 1 and 2 means the curve is closed, condition 3 means it doesn't have self-intersection or self-touching, and the last condition means the "thickness" of the curve is unitary.

Definition 5 (inside region and outside region of a picture)**.** Let $\boldsymbol{X} \in \{0,1\}^{M\times N}$ and suppose that $\mathtt{F}_{\boldsymbol{X}}$ (the set of pixels in $\boldsymbol{X}$ with value 1) is a digital Jordan curve in $\boldsymbol{X}$. We define the outside region of $\boldsymbol{X}$ as the set of all pixels $\mathtt{v}_0 \in \boldsymbol{X}$ with value 0 such that there exists a sequence of 0-value pixels starting from $\mathtt{u}$,

$$(\mathtt{v}_0, \mathtt{v}_1, \mathtt{v}_2, \ldots, \mathtt{v}_L), \tag{10}$$

where pixels $\mathtt{v}_i$ and $\mathtt{v}_{i+1}$ have a common vertex, and $\mathtt{v}_L$ is the border of the image $\boldsymbol{X}$.

We define the inside region of $\boldsymbol{X}$ as the set of pixels with value 0 that are not in the outside region.

B Details for Proof of Theorem 1

Weights and biases are determined as follows: $\boldsymbol{w}^{(2)} = \boldsymbol{1}_{1\times N} \in \mathbb{R}^{1\times N}$, $\boldsymbol{B}_r^{(2)} \in \mathbb{R}^{M\times N}$ is defined by

$$(B_r^{(2)})_{m,n} = \begin{cases} -(2u-\frac{1}{2}) & \text{if } r = 3u+1, \\ -2u & \text{if } r = 3u+2, \\ -(2u+\frac{1}{2}) & \text{if } r = 3u+3, \end{cases} \tag{11}$$

for all $1 \le m \le M$ and $1 \le n \le N$ where $u \in \mathbb{N}_0$, $\boldsymbol{w}^{(3)} \in \mathbb{R}^{1\times 1\times C'}$ is defined by

$$w_r^{(3)} = \begin{cases} -2 & \text{if } r \equiv 1 \pmod 3, \\ 4 & \text{if } r \equiv 2 \pmod 3, \\ -2 & \text{if } r \equiv 3 \pmod 3, \end{cases} \tag{12}$$

and $\boldsymbol{B}^{(3)} = \boldsymbol{1}_{M\times N} \in \mathbb{R}^{M\times N}$.

We had $\boldsymbol{1}_{1\times N} \cdot \vec{H}_{m,n}^{(1)} = \vec{X}_{m,n} \cdot \vec{X}_{m+1,n}$. Therefore,

$$\mathrm{H}_{m,n,r}^{(2)} = \boldsymbol{1}_{1\times N} \cdot \vec{H}_{m,n}^{(1)} + (B_r^{(2)})_{m,n} = \vec{X}_{m,n} \cdot \vec{X}_{m+1,n} + (B_r^{(2)})_{m,n}. \tag{13}$$

Next, notice that for each $r = 3u+i$ with $i \in \{1,2,3\}$, there exists $v \in \mathbb{Z}$ such that it holds $\vec{X}_{m,n} \cdot \vec{X}_{m+1,n} = 2u+v$. Then,

$$\mathrm{H}_{m,n,r}^{(2)} = \begin{cases} \left[v+\frac{1}{2}\right]_+ & \text{if } r = 3u+1 \\ [v]_+ & \text{if } r = 3u+2 \\ \left[v-\frac{1}{2}\right]_+ & \text{if } r = 3u+3, \end{cases} \tag{14}$$

and summing these three equations we can see that

$$-2\mathrm{H}_{m,n,3u+1}^{(2)} + 4\mathrm{H}_{m,n,3u+2}^{(2)} - 2\mathrm{H}_{m,n,3u+3}^{(2)} = \begin{cases} -1 & \text{if } v = 0, \\ 0 & \text{otherwise.} \end{cases} \tag{15}$$

Lastly, by definition of the output layer we obtain

$$\begin{aligned} Y_{m,n} &= \left[\sum_{r=1}^{3\left(\lfloor \frac{C}{2} \rfloor + 1\right)} w_r^{(3)} \mathrm{H}_{m,n,r}^{(2)} + 1 \right]_+ \\ &= \left[\sum_{u=0}^{\lfloor \frac{C}{2} \rfloor} \left(w_{3u+1}^{(3)} \mathrm{H}_{m,n,3u+1}^{(2)} + w_{3u+2}^{(3)} \mathrm{H}_{m,n,3u+2}^{(2)} + w_{3u+3}^{(3)} \mathrm{H}_{m,n,3u+3}^{(2)} \right) + 1 \right]_+ \\ &= \left[\left(\sum_{u=0}^{\lfloor \frac{C}{2} \rfloor} -2\mathrm{H}_{m,n,3u+1}^{(2)} + 4\mathrm{H}_{m,n,3u+2}^{(2)} - 2\mathrm{H}_{m,n,3u+3}^{(2)} \right) + 1 \right]_+ \\ &= \begin{cases} 0 & \text{if } \vec{X}_{m,n} \cdot \vec{X}_{m+1,n} = 2u \quad \text{for some } 0 \le u \le \lfloor \frac{C}{2} \rfloor \\ 1 & \text{otherwise.} \end{cases} \end{aligned} \tag{16}$$

By *Lemma 1* and the fact that $\vec{X}_{m,n} \cdot \vec{X}_{m+1,n} \le C$, we conclude that

$$Y_{m,n} = \begin{cases} 0 & \text{if } \vec{X}_{m,n} \cdot \vec{X}_{m+1,n} \text{ is even} \quad \Leftrightarrow X_{m,n} \text{ is outside,} \\ 1 & \text{if } \vec{X}_{m,n} \cdot \vec{X}_{m+1,n} \text{ is odd} \quad \Leftrightarrow X_{m,n} \text{ is inside.} \end{cases} \tag{17}$$

□

C Neural Network Representations of Boolean Operations

In the Coloring Network, we use neural network representations (implementations) of Boolean NOT and Boolean AND as building blocks. Here we explain the details.

C.1 Neural Network Representation of Boolean NOT

In addition to the normal notation $\neg$, we use $\mathrm{NOT} : \{0,1\} \to \{0,1\}$ to denote the Boolean NOT. That is,

$$\mathrm{NOT}(x) = \neg x = \begin{cases} 1 & \text{if } x = 0, \\ 0 & \text{if } x = 1. \end{cases} \tag{18}$$

Like other functions, we use $\mathbf{NOT}$ to denote element-wise application of NOT to a vector or a matrix. We here show that this function can be represented by a neural network with rectified linear function.

Let $x \in \{0,1\}$ and consider the neural network $f^{\mathrm{N}}(\cdot;\boldsymbol{\theta})$ defined by

$$f^{\mathrm{N}}(x;\boldsymbol{\theta}) \coloneqq [wx + b]_+ \in \{0,1\}, \tag{19}$$

where $= (w,b) \in \mathbb{R}^2$. It is easy to check that defining $\boldsymbol{\theta} = (-1,1)$, we obtain

$$f^{\mathrm{N}}(x;\boldsymbol{\theta}_{\mathbf{N}}) = NOT(x).$$

We can then generalize to higher-order cases as follows. Let $\boldsymbol{X} \in \{0,1\}^{I\times J}$ and consider the neural network $\boldsymbol{F}^{\mathrm{N}}(\cdot;\boldsymbol{\Theta})$ defined by

$$\boldsymbol{F}^{\mathrm{N}}(\boldsymbol{X};\boldsymbol{\Theta}) \coloneqq [w * \boldsymbol{X} + \boldsymbol{B}]_+ \in \mathbb{R}^{I\times J},$$

where $\boldsymbol{\Theta} = (w, \boldsymbol{B})$. Setting $\boldsymbol{\Theta}_{\mathrm{N}} = (-1, \boldsymbol{1}_{I\times J})$, we obtain $\boldsymbol{F}^{\mathrm{N}}(\boldsymbol{X};\boldsymbol{\Theta}_{\mathrm{N}}) = \mathbf{NOT}(\boldsymbol{X})$.

C.2 Neural Network Representation of Boolean AND

In addition to the normal operator notation $\wedge$, we use $\mathrm{AND} : \{0,1\} \times \{0,1\} \to \{0,1\}$ to denote the Boolean AND. That is,

$$\mathrm{AND}(x_1,x_2) = x_1 \wedge x_2 = x_1 \cdot x_2 = \begin{cases} 1 & \text{if } x_1 = 1, x_2 = 1 \\ 0 & \text{otherwise.} \end{cases} \tag{20}$$

Here, $\mathbf{AND}$ denotes element-wise application of AND to vectors or matrices. This can also be implemented by a neural network with rectified linear activation function as shown below.

Let $x_1, x_2 \in \{0,1\}$ and consider the neural network $f^{\mathrm{A}}(\cdot,\cdot;\boldsymbol{\theta})$ defined by

$$y = f^{\mathrm{A}}(x_1,x_2;\boldsymbol{\theta}) \coloneqq [\boldsymbol{w}\cdot[x_1,x_2]^\top + b]_+ \in \{0,1\},$$

where $\boldsymbol{\theta} = (\boldsymbol{w}, b)$. It is easy to verify that setting $\boldsymbol{\theta}_{\mathrm{A}} = (\boldsymbol{1}_2, -1)$ we obtain

$$f^{\mathrm{A}}(x_1,x_2;\boldsymbol{\theta}_{\mathrm{A}}) = \mathrm{AND}(x_1,x_2)$$

As in the case of the Boolean NOT, we can generalize this function to higher order cases, but before doing so, we introduce some notation.

Informally speaking, we use $\boldsymbol{A} \boxplus \boldsymbol{B}$ to denote a tensor made by "stacking" a matrix $\boldsymbol{B}$ "behind" a matrix $\boldsymbol{A}$. Formal definition is as follows. Consider $\boldsymbol{A} = [A_{i,j}] \in \mathbb{R}^{I\times J}$ and $\boldsymbol{B} = [B_{i,j}] \in \mathbb{R}^{I\times J}$. We define a tensor $\boldsymbol{A} \boxplus \boldsymbol{B} = [(\boldsymbol{A} \boxplus \boldsymbol{B})_{i,j,k}] \in \mathbb{R}^{I\times J\times 2}$ by the following equations:

$$(\boldsymbol{A} \boxplus \boldsymbol{B})_{i,j,1} \coloneqq A_{i,j}, \tag{21}$$

$$(\boldsymbol{A} \boxplus \boldsymbol{B})_{i,j,2} \coloneqq B_{i,j}, \tag{22}$$

for all $1 \le i \le I$ and $1 \le j \le J$.

We can then extend the neural network f^{A} as follows. Let $\boldsymbol{X}_1, \boldsymbol{X}_2 \in \{0,1\}^{I \times J}$ and consider the neural network $\boldsymbol{F}^{\mathrm{A}}(\cdot, \cdot; \boldsymbol{\Theta})$ defined by

$$\boldsymbol{F}^{\mathrm{A}}(\boldsymbol{X}_1, \boldsymbol{X}_2; \boldsymbol{\Theta}) \coloneqq [\boldsymbol{w} * (\boldsymbol{X}_1 \boxplus \boldsymbol{X}_2) + \boldsymbol{B}]_+ \in \mathbb{R}^{I \times J},$$

where $\boldsymbol{\Theta} = (\boldsymbol{w}, \boldsymbol{B})$. Setting $\boldsymbol{\Theta}_{\mathrm{A}} = ([1,1], -\boldsymbol{1}_{I \times J})$ we obtain

$$\boldsymbol{F}^{\mathrm{A}}(\boldsymbol{X}_1, \boldsymbol{X}_2; \boldsymbol{\Theta}_{\mathrm{A}}) = \mathbf{AND}(\boldsymbol{X}_1, \boldsymbol{X}_2) = \boldsymbol{X}_1 \odot \boldsymbol{X}_2.$$

where $\odot$ expresses the Hadamard porduct (element-wise product) of two matrices.